Internet Engineering Task Force
INTERNET DRAFT
February 8, 2000
Expires August 8, 2000
<draft-ietf-megaco-protocol-06.txt>

Fernando Cuervo
Nortel Networks
Bryan Hill
Gotham Networks
Nancy Greene
Nortel Networks
Christian Huitema
Telcordia Technologies
Abdallah Rayhan
Nortel Networks
Brian Rosen
Marconi
John Segers
Lucent Technologies

Megaco Protocol

Status of this document

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at
http://www.ietf.org/shadow.html.

This document will expire in August 2000.

1. SCOPE 7
2. REFERENCES 7
2.1. Normative references 7

2.2. Informative references 8
3. DEFINITIONS 9
4. ABBREVIATIONS 10
5. CONVENTIONS 10
6. CONNECTION MODEL 10
6.1. Contexts 14
6.1.1. Context Attributes and Descriptors 15
6.1.2. Creating, Deleting and Modifying Contexts 15
6.2. Terminations 15
6.2.1. Termination Dynamics 16
6.2.2. TerminationIDs 16
6.2.3. Packages 17
6.2.4. Termination Properties and Descriptors 17
6.2.5. Root Termination 19
7. COMMANDS 20
7.1. Descriptors 21
7.1.1. Specifying Parameters 21
7.1.2. Modem Descriptor 22
7.1.3. Multiplex Descriptor 22
7.1.4. Media Descriptor 22
7.1.5. Termination State Descriptor 23
7.1.6. Stream Descriptor 24
7.1.7. LocalControl Descriptor 24
7.1.8. Local and Remote Descriptors 25
7.1.9. Events Descriptor 28
7.1.10. EventBuffer Descriptor 29
7.1.11. Signals Descriptor 29
7.1.12. Audit Descriptor 31
7.1.13. ServiceChange Descriptor 32
7.1.14. DigitMap Descriptor 32
7.1.15. Statistics Descriptor 35
7.1.16. Packages Descriptor 36
7.1.17. ObservedEvents Descriptor 36
7.1.18. Topology Descriptor 36
7.2. Command Application Programming Interface 39
7.2.1. Add 39
7.2.2. Modify 41
7.2.3. Subtract 42
7.2.4. Move 43
7.2.5. AuditValue 44
7.2.6. AuditCapabilities 45
7.2.7. Notify 46
7.2.8. ServiceChange 46
7.2.9. Manipulating and Auditing Context Attributes 50
7.2.10. Generic Command Syntax 51

7.3. Command Error Codes 51
8. TRANSACTIONS 53
8.1. Common Parameters 54
8.1.1. Transaction Identifiers 54
8.1.2. Context Identifiers 55
8.2. Transaction Application Programming Interface 55

8.2.1. TransactionRequest 55
8.2.2. TransactionReply 56
8.2.3. TransactionPending 57
8.3. Messages 57
9. TRANSPORT 58
9.1. Ordering of Commands 58
9.2. Protection against Restart Avalanche 59
10. SECURITY CONSIDERATIONS 60
10.1. Protection of Protocol Connections 60
10.2. Interim AH scheme 61
10.3. Protection of Media Connections 62
11. MG-MGC CONTROL INTERFACE 63
11.1. Multiple Virtual MGs 63
11.2. Cold Start 64
11.3. Negotiation of Protocol Version 64
11.4. Failure of an MG 65
11.5. Failure of an MGC 65
12. PACKAGE DEFINITION 66
12.1. Guidelines for defining packages 66
12.1.1. Package Overall description of the package, ... 67
12.1.2. Properties 67
12.1.3. Events 68
12.1.4. Signals 68
12.1.5. Statistics 69
12.1.6. Procedures 69
12.2. Guidelines to defining Properties, Statistics and ... 69
12.3. Lists 69
12.4. Identifiers 69
12.5. Package Registration 70
13. IANA CONSIDERATIONS 70
13.1. Packages 70
13.2. Error Codes 70
13.3. ServiceChange Reasons 71
14. CONTACT INFORMATION 71
ANNEX A BINARY ENCODING OF THE PROTOCOL (NORMATIVE) 73
A.1. Coding of wildcards 73
A.2. ASN.1 syntax specification 74
A.3. Digit maps and path names 89
ANNEX B TEXT ENCODING OF THE PROTOCOL (NORMATIVE) 90
B.1. Coding of wildcards 90
B.2. ABNF specification 90
ANNEX C TAGS FOR MEDIA STREAM PROPERTIES (NORMATIVE) 101

C.1. General Media Attributes 102
C.2. Mux Properties 102
C.3. General bearer properties 102
C.4. General ATM properties 103
C.5. Frame Relay 104
C.6. IP 104
C.7. ATM AAL2 104
C.8. ATM AAL1 104
C.9. Bearer Capabilities 105
C.10. AAL5 Properties 106

C.11. SDP Equivalents 107
C.12. H.245 107
ANNEX D TRANSPORT OVER IP (NORMATIVE) 107
D.1. Transport over IP/UDP using Application Level 107
D.1.1. Providing At-Most-Once Functionality 107
D.1.2. Transaction identifiers and three-way handshake 108
D.1.3. Computing retransmission timers 110
D.1.4. Provisional responses 111
D.1.5. Repeating Requests, Responses and 111
D.2. Using TCP 112
D.2.1. Providing the At-Most-Once functionality 113
D.2.2. Transaction identifiers and three way handshake 113
D.2.3. Computing retransmission timers 113
D.2.4. Provisional responses 113
D.2.5. Ordering of commands 114
ANNEX E BASIC PACKAGES 114
E.1. Generic 114
E.1.1. Properties 114
E.1.2. Events 114
E.2.2. Events 118
E.2.3. Signals 118
E.2.4. Statistics 118
E.2.5. Procedures 118
E.3. Tone Generator Package 118
E.3.1. Properties 118
E.3.2. Events 118
E.3.3. Signals 118
E.3.4. Statistics 119
E.3.5. Procedures 119
E.4. Tone Detection Package 119
E.4.1. Properties 119
E.4.2. Events 119
E.4.3. Signals 121
E.4.4. Statistics 121
E.4.5. Procedures 121
E.5. Basic DTMF Generator Package 121
E.5.1. Properties 121
E.5.2. Events 121

E.5.3. Signals 122
E.5.4. Statistics 122
E.5.5. Procedures 122
E.6. DTMF detection Package 123
E.6.1. Properties 123
E.6.2. Events 123
E.6.3. Signals 125
E.6.4. Statistics 125
E.6.5. Procedures 125
E.7. Call Progress Tones Generator Package 125
E.7.1. Properties 125
E.7.2. Events 125
E.7.3. Signals 125

E.7.4. Statistics 126
E.7.5. Procedures 126
E.8. Call Progress Tones Detection Package 126
E.8.1. Properties 126
E.8.2. Events 126
E.8.3. Signals 127
E.8.4. Statistics 127
E.8.5. Procedures 127
E.9. Analog Line Supervision Package 127
E.9.1. Properties 127
E.9.2. Events 127
E.9.3. Signals 128
E.9.4. Statistics 129
E.9.5. Procedures 129
E.10. Basic Continuity Package 129
E.10.1. Properties 129
E.10.2. Events 129
E.10.3. Signals 129
E.10.4. Statistics 130
E.10.5. Procedures 130
E.11. Network Package 130
E.11.1. Properties 131
E.11.2. Events 131
E.11.3. Signals 132
E.11.4. Statistics 132
E.11.5. Procedures 132
E.12. RTP Package 132
E.12.1. Properties 133
E.12.2. Events 133
E.12.3. Signals 133
E.12.4. Statistics 133
E.12.5. Procedures 134
E.13. DS0 Package 134
E.13.1. Properties 134
E.13.2. Events 135

E.13.3. Signals 131
E.13.4. Statistics 131
E.13.5. Procedures 132
APPENDIX A EXAMPLE CALL FLOWS (INFORMATIVE) 132
A.1. Residential Gateway to Residential Gateway Call 132
A.1.1. Programming Residential GW Analog Line 132

TABLE OF FIGURES

Figure 1: Example of MEGACOH.248 Connection Model 12
Figure 3: Example Call Waiting Scenario / Alerting Applied to T1 13
Figure 4: Example Call Waiting Scenario / Answer by T1 14
Figure 5: Example topologies 38
Figure 6: Transactions, Actions and Commands.....53

1. SCOPE

MEGACO defines the protocols used between elements of a physically decomposed multimedia Gateway consisting of a Media Gateway and a Media Gateway Controller. There are no functional differences from a system view between a decomposed gateway, with distributed sub- components potentially on more than one physical device, and a monolithic gateway. This document does not define how gateways, multipoint control units or integrated voice response units (IVRs) work. Instead it creates a general framework that is suitable for these applications.

Packet network interfaces may include IP, ATM or possibly others. The interfaces will support a variety of SCN signalling systems, including tone signalling, ISDN, ISUP, QSIG, and GSM. National variants of these signalling systems will be supported where applicable.

The protocol definition in this document is common text with ITU Recommendation H.248.

2. REFERENCES

2.1. Normative references

ITU-T Recommendation H.225.0 (1998): "Call Signalling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems".

ITU-T Recommendation H.235 (02/98): "Security and encryption for H-Series (H.323 and other H.245-based) multimedia terminals".

ITU-T Recommendation H.245 (1998): "Control Protocol for Multimedia Communication".

ITU-T Recommendation H.323 (1998): "Packet Based Multimedia Communication Systems".

ITU-T Recommendation I.363.1 (08/96), "B-ISDN ATM Adaptation Layer specification: Type 1 AAL".

ITU-T Recommendation I.366.1 (06/98), "Segmentation and Reassembly Service Specific Convergence Sublayer for the AAL type 2".

ITU-T Recommendation I.366.2 (02/99), "AAL type 2 service specific convergence sublayer for trunking".

ITU-T Recommendation Q.931 (05/98): "Digital Subscriber Signalling System No. 1 (DSS 1) - ISDN User-Network Interface Layer 3 Specification for Basic Call Control"

ITU-T Recommendation X.680 (1997): "Information technology-Abstract Syntax Notation One (ASN.1): Specification of basic notation".

ITU-T draft Recommendation H.246 (1998), "Interworking of H-series multimedia terminals with H-series multimedia terminals and voice/voiceband terminals on GSTN and ISDN".

RFC 1006, "ISO Transport Service on top of the TCP, Version 3", Marshall T. Rose, Dwight E. Cass, May 1987.

RFC 2119, "Key words for use in RFCs to Indicate Requirement Levels", Scott Bradner, March 1997. RFC 2234, "Augmented BNF for Syntax Specifi-

cations: ABNF", D. Crocker, P. Overell, November 1997.

RFC 2327, "SDP: Session Description Protocol", M. Handley, V. Jacobson, April 1998.

RFC 2402, "IP Authentication Header", S. Kent, R. Atkinson, November 1998.

RFC 2406, "IP Encapsulating Security Payload (ESP)", S. Kent, R. Atkinson, November 1998.

2.2. Informative references

ITU-T Recommendation E.180/Q.35 (1998): "Technical characteristics of tones for the telephone service"

ITU-T Recommendation Q.724 (1988): "Signalling procedures"

RFC 768, "User Datagram Protocol", J.Postel, August 1980.

RFC 793, "TRANSMISSION CONTROL PROTOCOL", J.Postel, September 1981.

RFC 1889, "RTP: A Transport Protocol for Real-Time Applications", H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, January 1996.

RFC 1890, "RTP Profile for Audio and Video Conferences with Minimal

Control", H. Schulzrinne, January 1996.

RFC 2401, "Security Architecture for the Internet Protocol", S. Kent, R. Atkinson, November 1998.

RFC 2543, " SIP: Session Initiation Protocol", M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, March 1999.

RFC 2460, "Internet Protocol, Version 6 (IPv6) Specification", S. Deering, R. Hinden, December 1998.

3. DEFINITIONS

Access Gateway: A type of gateway that provides a User to Network Interface (UNI) such as ISDN.

Descriptor: A syntactic element of the protocol that groups related properties. For instance, the properties of a media flow on the MG can be set by the MGC by including the appropriate descriptor in a command.

Media Gateway (MG): The media gateway converts media provided in one type of network to the format required in another type of network. For example, a MG could terminate bearer channels from a switched circuit network (i.e., DS0s) and media streams from a packet network (e.g., RTP streams in an IP network). This gateway may be capable of processing

audio, video and T.120 alone or in any combination, and will be capable of full duplex media translations. The MG may also play audio/video messages and performs other IVR functions, or may perform media conferencing.

Media Gateway Controller (MGC): Controls the parts of the call state that pertain to connection control for media channels in a MG.

Multipoint Control Unit (MCU): An entity that controls the setup and coordination of a multi- user conference that typically includes processing of audio, video and data.

Residential Gateway: A gateway that interworks an analogue line to a packet network. A residential gateway typically contains one or two analogue lines and is located at the customer premises.

SCN FAS Signalling Gateway: This function contains the SCN Signalling Interface that terminates SS7, ISDN or other signalling links where the call control channel and bearer channels are collocated in the same physical span.

SCN NFAS Signalling Gateway: This function contains the SCN Signalling Interface that terminates SS7 or other signalling links where the call

control channels are separated from bearer channels.

Stream: Bidirectional media or control flow received/sent by a media gateway as part of a call or conference.

Trunk: A communication channel between two switching systems such as a DS0 on a T1 or E1 line.

Trunking Gateway: A gateway between SCN network and packet network that typically terminates a large number of digital circuits.

4. ABBREVIATIONS

This recommendation defines the following terms.

ATM	Asynchronous Transfer Mode
BRI	Basic Rate Interface
CAS	Channel Associated Signalling
DTMF	Dual Tone Multi-Frequency
FAS	Facility Associated Signalling
GW	GateWay
IP	Internet Protocol
ISUP	ISDN User Part
MG	Media Gateway
MGC	Media Gateway Controller
NFAS	Non-Facility Associated Signalling
PRI	Primary Rate Interface
PSTN	Public Switched Telephone Network
QoS	Quality of Service

RTP	Real-time Transport Protocol
SCN	Switched Circuit Network
SG	Signalling Gateway
SS7	Signalling System No. 7

5. CONVENTIONS

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

6. CONNECTION MODEL

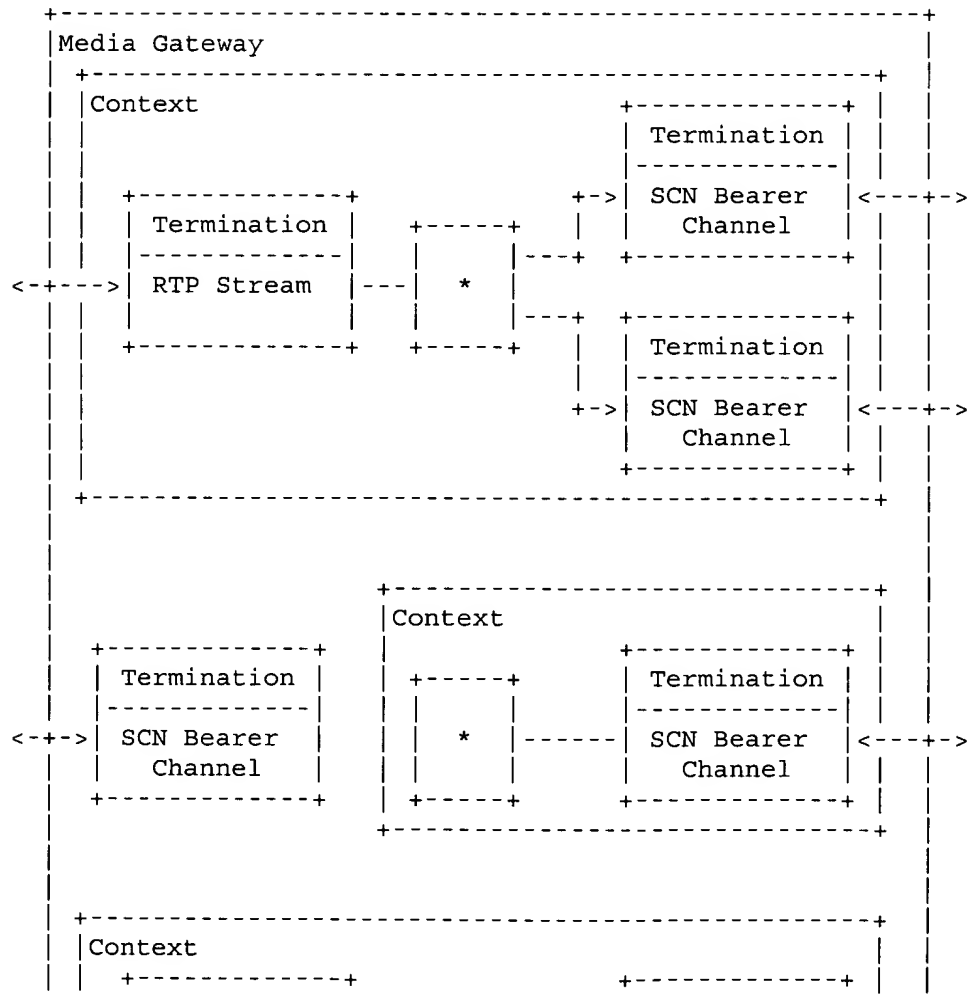
The connection model for the protocol describes the logical entities, or objects, within the Media Gateway that can be controlled by the Media Gateway Controller. The main abstractions used in the connection model are Terminations and Contexts.

A Termination sources and/or sinks one or more streams. In a multimedia conference, a Termination can be multimedia and sources or sinks

multiple media streams. The media stream parameters, as well as modem, and bearer parameters are encapsulated within the Termination.

A Context is an association between a collection of Terminations. There is a special type of Context, the null Context, which contains all Terminations that are not associated to any other Termination. For instance, in a decomposed access gateway, all idle lines are represented by Terminations in the null Context.

Following is a graphical depiction of these concepts. The diagram of Figure 1 gives several examples and is not meant to be an all-inclusive illustration. The asterisk box in each of the Contexts represents the logical association of Terminations implied by the Context.


Media Gateway
Context
Termination
SCN Bearer Channel
Termination
RTP Stream
*
Termination
SCN Bearer Channel
Context
Termination
SCN Bearer Channel
*
Termination
SCN Bearer Channel
Context
Termination
SCN Bearer Channel

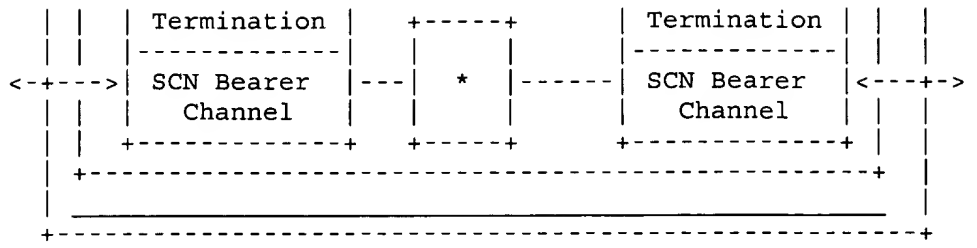


Figure 1: Example of MEGACO Connection Model

The example below shows an example of one way to accomplish a call-waiting scenario in a decomposed access gateway, illustrating the relocation of a Termination between Contexts. Terminations T1 and T2 belong to Context C1 in a two-way audio call. A second audio call is waiting

for T1 from Termination T3. T3 is alone in Context C2. T1 accepts the call from T3, placing T2 on hold. This action results in T1 moving into Context C2, as shown below.

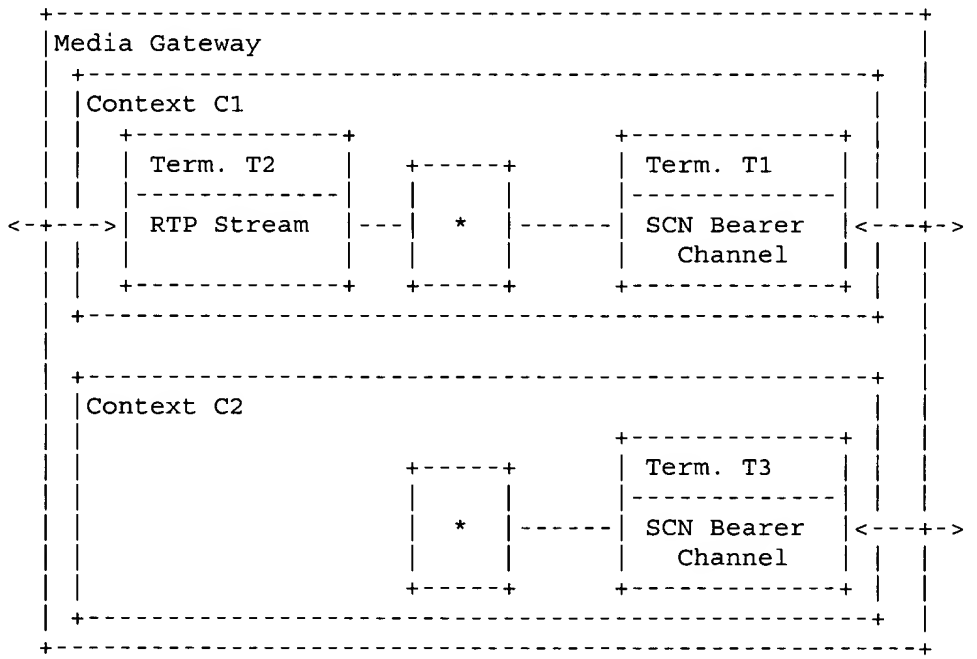


Figure 2 Example Call Waiting Scenario / Alerting Applied to T1

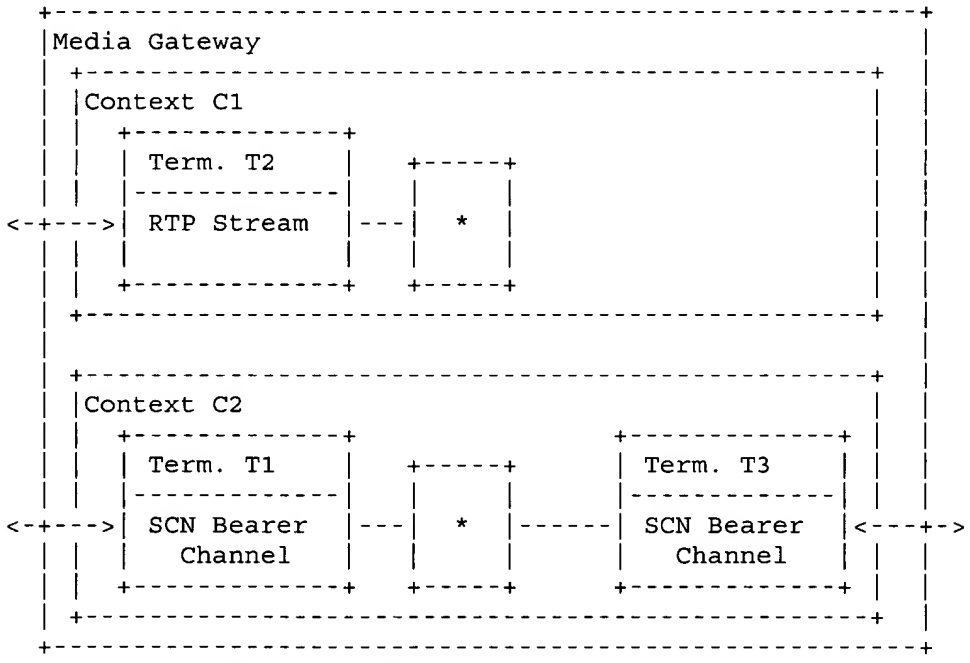


Figure 3. Example Call Waiting Scenario / Answer by T1

6.1. Contexts

A Context is an association between a number of Terminations. The Context describes the topology (who hears/sees whom) and the media mixing and/or switching parameters if more than two Terminations are involved in the association.

There is a special Context called the null Context. It contains Terminations that are not associated to any other Termination. Terminations in the null Context can have their parameters examined or modified, and may have events detected on them.

In general, an Add command is used to add Terminations to Contexts. If the MGC does not specify an existing Context to which the Termination is

to be added, the MG creates a new Context. A Termination may be removed from a Context with a Subtract command, and a Termination may be moved from one Context to another with a Move command. A Termination SHALL exist in only one Context at a time.

The maximum number of Terminations in a Context is a MG property. Media gateways that offer only point-to-point connectivity might allow at most two Terminations per Context. Media gateways that support multipoint

conferences might allow three or more terminations per Context.

6.1.1. Context Attributes and Descriptors

The attributes of Contexts are:

ContextID, a 32 bit unsigned integer chosen by the MG. The topology (who hears/sees whom). The topology of a Context describes the flow of media between the Terminations within a Context. In contrast, the mode of a Termination (send/receive/...) describes the flow of the media at the ingress/egress of the media gateway.

The priority is used for a context in order to provide the MG with information about a certain precedence handling for a context. The MGC can also use the priority to control autonomously the traffic precedence in the MG in a smooth way in certain situations (e.g. restart), when a lot of contexts must be handled simultaneously.

* An indicator for an emergency call is also provided to allow a preference handling in the MG.

6.1.2. Creating, Deleting and Modifying Contexts

The protocol can be used to (implicitly) create Contexts and modify the parameter values of existing Contexts. The protocol has commands to add Terminations to Contexts, subtract them from Contexts, and to move Terminations between Contexts. Contexts are deleted implicitly when the last remaining Termination is subtracted or moved out.

6.2. Terminations

A Termination is a logical entity on a MG that sources and/or sinks media and/or control streams. A Termination is described by a number of characterizing Properties, which are grouped in a set of Descriptors that are included in commands. Terminations have unique identities (TerminationIDs), assigned by the MG at the time of their creation.

Terminations representing physical entities have a semi-permanent existence. For example, a Termination representing a TDM channel might exist for as long as it is provisioned in the gateway. Terminations representing ephemeral information flows, such as RTP flows, would usually exist only for the duration of their use.

Ephemeral Terminations are created by means of an Add command. They are destroyed by means of a Subtract command. In contrast, when a physical Termination is Added to or Subtracted from a Context, it is taken from or to the null Context, respectively.

Terminations may have signals applied to them. Signals are MG generated media streams such as tones and announcements as well as line signals such as hookswitch. Terminations may be programmed to detect Events, the occurrence of which can trigger notification messages to the MGC, or action by the MG. Statistics may be accumulated on a Termination. Statistics are reported to the MGC upon request (by means of the Audit-Value command, see section 7.2.5) and when the Termination is taken out of the call it is in.

Multimedia gateways may process multiplexed media streams. For example, Recommendation H.221 describes a frame structure for multiple media streams multiplexed on a number of digital 64 kbit/s channels. Such a case is handled in the connection model in the following way. For every bearer channel that carries part of the multiplexed streams, there is a Termination. The Terminations that source/sink the digital channels are connected to a separate Termination called the multiplexing Termination. This Termination describes the multiplex used (e.g. how the H.221 frames are carried over the digital channels used). The MuxDescriptor is used to this end. If multiple media are carried, this Termination contains multiple StreamDescriptors. The media streams can be associated with streams sourced/sunk by other Terminations in the Context.

Terminations may be created which represent multiplexed bearers, such as an ATM AAL2. When a new multiplexed bearer is to be created, an ephemeral termination is created in a context established for this purpose. When the termination is subtracted, the multiplexed bearer is destroyed.

6.2.1. Termination Dynamics

The protocol can be used to create new Terminations and to modify property values of existing Terminations. These modifications include the possibility of adding or removing events and/or signals. The Termination properties, and events and signals are described in the ensuing sections. An MGC can only release/modify terminations and the resources that the termination represents which it has previously seized via, e.g., the Add command.

6.2.2. TerminationIDs

Terminations are referenced by a TerminationID, which is an arbitrary schema chosen by the MG.

TerminationIDs of physical Terminations are provisioned in the Media Gateway. The TerminationIDs may be chosen to have structure. For instance, a TerminationID may consist of trunk group and a trunk within the group.

A wildcarding mechanism using two types of wildcards can be used with

TerminationIDs. The two wildcards are ALL and CHOOSE. The former is used to address multiple Terminations at once, while the latter is used to indicate to a media gateway that it must select a Termination satisfying the partially specified TerminationID. This allows, for instance, that a MGC instructs a MG to choose a circuit within a trunk group.

When ALL is used in the TerminationID of a command, the effect is identical to repeating the command with each of the matching TerminationIDs. Since each of these commands may generate a response, the size of the entire response may be large. If individual responses are not required, a wildcard response may be requested. In such a case, a single response is generated, which contains the UNION of all of the individual responses which otherwise would have been generated. Wildcard response may be particularly useful in the Audit commands.

The encoding of the wildcarding mechanism is detailed in Annexes A and B.

6.2.3. Packages

Different types of gateways may implement Terminations that have widely differing characteristics. Variations in Terminations are accommodated in the protocol by allowing Terminations to have optional Properties, Events, Signals and Statistics implemented by MGs.

In order to achieve MG/MGC interoperability, such options are grouped into Packages, and a Termination realizes a set of such Packages. More information on definition of packages can be found in section 12. An MGC can audit a Termination to determine which Packages it realizes.

Properties, Events, Signals and Statistics defined in Packages, as well as parameters to them, are referenced by identifiers (Ids). Identifiers are scoped. For each package, PropertyIds, EventIds, SignalIds, StatisticsIds and ParameterIds have unique name spaces and the same identifier may be used in each of them. Two PropertyIds in different packages may also have the same identifier, etc.

6.2.4. Termination Properties and Descriptors

Terminations have properties. The properties have unique PropertyIDs. Most properties have default values. When a Termination is created, properties get their default values, unless the controller specifically sets a different value. The default value of a property of a physical Termination can be changed by setting it to a different value when the Termination is in the null Context. Every time such a Termination returns to the null Context, the values of its properties are reset to this default value.

There are a number of common properties for Terminations and properties specific to media streams. The common properties are also called the termination state properties. For each media stream, there are local properties and properties of the received and transmitted flows.

Properties not included in the base protocol are defined in Packages. These properties are referred to by a name consisting of the PackageName and a PropertyId. Most properties have default values described in the Package description. Properties may be read-only or read/write. The possible values of a property may be audited, as can their current values. For properties that are read/write, the MGC can set their values. A property may be declared as "Global" which has a single value shared by all terminations realizing the package. Related properties are grouped into descriptors for convenience.

When a Termination is Added to a Context, the value of its read/write properties can be set by including the appropriate descriptors as parameters to the Add command. Properties not mentioned in the command retain their prior values. Similarly, a property of a Termination in a Context may have its value changed by the Modify command. Properties not mentioned in the Modify command retain their prior values. Properties may also have their values changed when a Termination is moved from one Context to another as a result of a Move command. In some cases, descriptors are returned as output from a command. The following table lists all of the possible Descriptors and their use. Not all descriptors are legal as input or output parameters to every command. Descriptors

Descriptor Name	Description
Modem	Identifies modem type and properties when applicable
Mux	Describes multiplex type for multimedia terminations (e.g. H.221, H.223, H.225.0) and Terminations forming the input mux.
Media	A list of media stream specifications (see 7.1.4)
TerminationState	Properties of a Termination (which can be defined in Packages) that are not stream specific.
Stream	A list of remote/local/localControl descriptors for a single stream
Local	Contains properties that specify the media flows that MG receives from the remote entity.

Remote	Contains properties that specify the media flows that the MG sends to the remote entity.	
LocalControl	Contains properties (which can be defined in packages) that are of interest between the MG and the MGC	
Events	Describes events to be detected by the MG and what to do when an event is detected	
EventBuffer	Describes events to be detected by the MG when Event Buffering is active	
Signals	Describes signals and/or actions to be applied (e.g. Busy Tone) to the Terminations	
Audit	In Audit commands, identifies which information is desired	
Packages	In AuditValue, returns a list of Packages realized by a Termination	
DigitMap	Instructions for handling DTMF tones at the MG	
ServiceChange	In ServiceChange, what, why service change occurred, etc.	
ObservedEvents	In Notify or AuditValue, report of events observed	
Statistics	In Subtract and Audit, Report of Statistics kept on a Termination.	

6.2.5. Root Termination

Occasionally, a command must refer to the entire gateway, rather than a termination within it. A special TerminationID, "Root" is reserved for this purpose. Packages may be defined on Root. Root thus may have properties and events (signals are not appropriate for root). Accordingly, the root TerminationID may appear in:

* a Modify command - to change a property or set an event

* a Notify command - to report an event

* an AuditValue return - to examine the values of properties implemented on root

* an AuditCapability - to determine what properties of root are implemented

* a ServiceChange - to declare the gateway in or out of service Any other use of the root TerminationID is an error.

7. COMMANDS

The protocol provides commands for manipulating the logical entities of the protocol connection model, Contexts and Terminations. Commands provide control at the finest level of granularity supported by the protocol. For example, Commands exist to add Terminations to a Context, modify Terminations, subtract Terminations from a Context, and audit properties of Contexts or Terminations. Commands provide for complete control of the properties of Contexts and Terminations. This includes specifying which events a Termination is to report, which signals/actions are to be applied to a Termination and specifying the topology of a Context (who hears/sees whom).

Most commands are for the specific use of the Media Gateway Controller as command initiator in controlling Media Gateways as command responders. The exceptions are the Notify and ServiceChange commands: Notify is sent from Media Gateway to Media Gateway Controller, and ServiceChange may be sent by either entity. Below is an overview of the commands; they are explained in more detail in section 7.2.

1. Add. The Add command adds a termination to a context. The Add command on the first Termination in a Context is used to create a Context.

2. Modify. The Modify command modifies the properties, events and signals of a termination.

3. Subtract. The Subtract command disconnects a Termination from its Context and returns statistics on the Termination's participation in the Context. The Subtract command on the last Termination in a Context deletes the Context.

4. Move. The Move command atomically moves a Termination to another context.

5. AuditValue. The AuditValue command returns the current state of properties, events, signals and statistics of Terminations.

6. AuditCapabilities. The AuditCapabilities command returns all the possible values for Termination properties, events and signals allowed by the Media Gateway.

7. Notify. The Notify command allows the Media Gateway to inform the Media Gateway Controller of the occurrence of events in the Media Gateway.

8. ServiceChange. The ServiceChange Command allows the Media Gateway to notify the Media Gateway Controller that a Termination or group

of Terminations is about to be taken out of service or has just been returned to service. ServiceChange is also used by the MG to announce its availability to an MGC (registration), and to notify the MGC of impending or completed restart of the MG. The MGC may announce a handover to the MG by sending it a ServiceChange command. The MGC may also use ServiceChange to instruct the MG to take a Termination or group of Terminations in or out of service.

These commands are detailed in sections 7.2.1 through 7.2.8

7.1. Descriptors

The parameters to a command are termed Descriptors. A Descriptor consists of a name and a list of items. Some items may have values. Many Commands share common Descriptors. This subsection enumerates these Descriptors. Descriptors may be returned as output from a command. Parameters and parameter usage specific to a given Command type are described in the subsection that describes the Command.

7.1.1. Specifying Parameters

Command parameters are structured into a number of descriptors. In general, the text format of descriptors is
DescriptorName=<someID>{parm=value, parm=value....}

Parameters may be fully specified, over-specified or under-specified:

1. Fully specified parameters have a single, unambiguous value that the command initiator is instructing the command responder to use for the specified parameter.

2. Under-specified parameters, using the CHOOSE value, allow the command responder to choose any value it can support.

3. Over-specified parameters have a list of potential values. The list order specifies the command initiator's order of preference of selection. The command responder chooses one value from the offered list and returns that value to the command initiator.

Unspecified mandatory parameters (i.e. mandatory parameters not specified in a descriptor) result in the command responder retaining the previous value for that parameter. Unspecified optional parameters result in the command responder using the default value of the parameter. Whenever a parameter is underspecified or overspecified, the descriptor containing the value chosen by the responder is included as output from the command.

Each command specifies the TerminationId the command operates on. This

TerminationId may be "wildcarded". When the TerminationId of a command

is wildcarded, the effect shall be as if the command was repeated with each of the TerminationIds matched.

7.1.2. Modem Descriptor

The Modem descriptor specifies the modem type and parameters, if any, required for use in e.g. H.324 and text conversation. The descriptor includes the following modem types: V.18, V.22, V.22bis, V.32, V.32bis, V.34, V.90, V.91, Synchronous ISDN, and allows for extensions. By default, no modem descriptor is present in a Termination.

7.1.3. Multiplex Descriptor

In multimedia calls, a number of media streams are carried on a (possibly different) number of bearers. The multiplex descriptor associates the media and the bearers. The descriptor includes the multiplex type:

* H.221
* H.223,
* H.226,
* V.76,
* Possible Extensions

and a set of TerminationIDs representing the multiplexed inputs, in order. For example:

```
Mux = H.221{ MyT3/1/2, MyT3/2/13, MyT3/3/6, MyT3/21/22}
```

7.1.4. Media Descriptor

The Media Descriptor specifies the parameters for all the media streams. These parameters are structured into two descriptors, a Termination State Descriptor, which specifies the properties of a termination that are not stream dependent, and one or more Stream Descriptors each of which describes a single media stream.

A stream is identified by a StreamID. The StreamID is used to link the streams in a Context that belong together. Multiple streams exiting a termination shall be synchronized with each other. Within the Stream Descriptor, there are up to three subsidiary descriptors, LocalControl, Local, and Remote. The relationship between these descriptors is thus:


```
Media Descriptor
        TerminationStateDescriptor
        Stream Descriptor
                LocalControl Descriptor
```

Local Descriptor
Remote Descriptor

StreamIDs are numbered from 1 upward. As a convenience a LocalControl, Local, or Remote descriptor may be included in the Media Descriptor without an enclosing Stream descriptor. In this case, the StreamID is assumed to be 1.

7.1.5. Termination State Descriptor

The Termination State Descriptor contains the ServiceStates property, the EventBuffer flag and properties of a termination (defined in Packages) that are not stream specific.

The ServiceStates property describes the overall state of the termination (not stream-specific). A Termination can be in one of the following states: "test", "out of service", or "in service". The "test" state indicates that the termination is not used for normal traffic, but for testing. A Termination with state "test" cannot be seized for traffic. The state "out of service" indicates a fault in the termination and cannot be used for traffic. The state "in service" indicates that a termination can be used or is being used for normal traffic. "in service" is the default state.

Values assigned to Properties may be simple values (integer/string/enumeration) or may be underspecified, where more than one value is supplied and the MG may make a choice:

* Alternative Values - multiple values in a list, one of which must be selected

* Ranges - minimum and maximum values, any value between min and max must be selected, boundary values included

* Greater Than/Less Than - value must be greater/less than specified value

* CHOOSE Wildcard - the MG chooses from the allowed values for the property The EventBuffer flag specifies whether events are buffered following detection of an event in the Events Descriptor, or processed immediately. See section 7.1.9 for details.

7.1.6. Stream Descriptor

A Stream descriptor specifies the parameters of a single bi-directional stream. These parameters are structured into three descriptors: one that contains termination properties specific to a stream and one each for local and remote flows. The Stream Descriptor includes a StreamID which identifies the stream. Streams are created by specifiying a new

StreamID on one of the terminations in a Context. A stream is deleted by setting empty Local and Remote descriptors for the stream with Reserve in LocalControl set to "false" on all terminations in the context which previously supported that stream.

If a termination is moved from one context to another, the following applies:

* if a streamID of an active stream in the moved termination matches a streamID in the context it was moved to, the associated stream remains active on that termination;

* if a streamID of an active stream in the moved termination does not match any streamID in the context it was moved to, the stream SHALL be set to inactive;

* if a stream is inactive on the moved termination, it SHALL remain inactive in the new context until its mode is changed explicitly;

* the modes of streams on terminations already present in the new context are unaffected by the fact that a termination is moved into the context.

7.1.7. LocalControl Descriptor

The LocalControl Descriptor contains the Mode property, the Reserve property and properties of a termination (defined in Packages) that are stream specific, and are of interest between the MG and the MGC. Values of properties may be underspecified as in section 7.1.5

The allowed values for the mode property are send-only, receive-only, send/receive, inactive, and loop-back. "Send" and "receive" are with respect to the exterior of the context, so that, for example, a stream set to mode=sendonly does not pass received media into the context. Signals and Events are not affected by mode.

The boolean-valued Reserve property of a Termination indicates what the MG is expected to do when it receives a local and/or remote descriptor.

If the value of Reserve is True, the MG SHALL reserve resources for all alternatives specified in the local and/or remote descriptors it

currently has resources available for. It SHALL respond with the alternatives it reserves resources for. If it cannot not support any of the alternatives, it SHALL respond with a reply to the MGC that contains empty local and/or remote descriptors.

If the value of Reserve is False, the MG SHALL choose one of the alternatives specified in the local descriptor (if present) and one of the alternatives specified in the remote descriptor (if present). If the MG has not yet reserved resources to support the selected alternative, it SHALL reserve the resources. If, on the other hand, it already reserved

resources for the Termination addressed (because of a prior exchange with Reserve equal to True), it SHALL release any excess resources it reserved previously. Finally, the MG shall send a reply to the MGC containing the alternatives for the local and/or remote descriptor that it selected. If the MG does not have sufficient resources to support any of the alternatives specified, is SHALL respond with error 510 (insufficient resources).

The default value of Reserve is False.

A new setting of the LocalControl Descriptor completely replaces the previous setting of that descriptor in the MG. Thus to retain information from the previous setting the MGC must include that information in the new setting. If the MGC wishes to delete some information from the existing descriptor, it merely resends the descriptor (in a Modify command) with the unwanted information stripped out

7.1.8. Local and Remote Descriptors

The MGC uses Local and Remote descriptors to reserve and commit MG resources for media decoding and encoding for the given Stream(s) and Termination to which they apply. The MG includes these descriptors in its response to indicate what it is actually prepared to support. The MG SHALL include additional properties and their values in its response if these properties are mandatory yet not present in the requests made by the MGC (e.g., by specifying detailed video encoding parameters where the MGC only specified the payload type).

Local refers to the media received by the MG and Remote refers to the media sent by the MG.

When text encoding the protocol, the descriptors consist of session descriptions as defined in SDP (RFC2327), except that the "s=", "t=" and "o=" lines are optional. When multiple session descriptions are provided in one descriptor, the "v=" lines are required as delimiters; otherwise they are optional. Implementations shall accept session descriptions that are fully conformant to RFC2327. When binary encoding the protocol the descriptor consists of groups of properties (tag-value pairs) as

specified in Annex C. Each such group may contain the parameters of a session description.

Below, the semantics of the local and remote descriptors are specified in detail. The specification consists of two parts. The first part specifies the interpretation of the contents of the descriptor. The second part specifies the actions the MG must take upon receiving the local and remote descriptors. The actions to taken by the MG depend on the value of the Reserve property of the LocalControl descriptor.

Either the local or the remote descriptor or both may be

* unspecified (i.e., absent),

* empty,

* underspecified through use of CHOOSE in a property value,

* fully specified, or

* overspecified through presentation of multiple groups of properties.

Where the descriptors have been passed from the MGC to the MG, they are interpreted according to the rules given in section 7.1.1, with the following additional comments for clarification:

a) An unspecified Local or Remote descriptor is considered to be a missing mandatory parameter. It requires the MG to use whatever was last specified for that descriptor. It is possible that there was no previously-specified value, in which case the descriptor concerned is ignored in further processing of the command.

b) An empty Local (Remote) descriptor in a message from the MGC signifies a request to release any resources reserved for the media flow received (sent).

c) If multiple groups of properties are present in a Local or Remote descriptor, the order of preference is descending.

d) Underspecified or overspecified properties within a group of properties sent by the MGC are requests for the MG to choose a value which it can support for each of those properties. In case of an overspecified property, the list of values is in descending order of preference.

Subject to the above rules, subsequent action depends on the value of the "Reserve" parameter in LocalControl.

If Reserve is true, the MG reserves the resources required to support any of the alternatives that it can currently support.

NOTE - If a Local or Remote descriptor contains multiple groups of properties, the MG is requested to reserve resources so that it can decode or encode one media stream according to any of the alternatives. For instance, if the Local descriptor contains two groups of properties, one specifying packetized G.711 A-law audio and the other G.723.1 audio, the MG reserves resources so that it can decode one audio stream encoded in G.711 A-law format or G.723.1 format. The MG should not reserve resources to decode two audio streams, one encoded in G.711 A-law and one in G.723.1.

* If the MG has insufficient resources to support all alternatives requested by the MGC and the MGC requested resources in both Local and Remote, the MGC should reserve resources to support at least

one alternative each within Local and Remote.

* If the MG has insufficient resources to support at least one alternative within a Local (Remote) descriptor received from the MGC, it shall return an empty Local (Remote) in response.

* In its response to the MGC, the MG SHALL include local and remote descriptors for all groups of properties it reserved resources for. If the MG is incapable of supporting at least one of the alternatives within the Local (Remote) descriptor received from the MGC, it SHALL return an empty Local (Remote) descriptor.

* If the Mode property of the TerminationState descriptor is RecvOnly or SendRecv, the MG must be prepared to receive media encoded according to any of the alternatives included in its response to the MGC.

If Reserve is False then the MG SHOULD apply the following rules to resolve Local and Remote to a single alternative each:

* If symmetric coding is not possible, the MG chooses the first alternative in Local for which it is able to support at least one alternative in Remote.

* If the MG is unable to support at least one Local and one Remote alternative, it returns Error 510 (Insufficient Resources).

* The MG returns its selected alternative in Local and Remote.

A new setting of a Local or Remote Descriptor completely replaces the previous setting of that descriptor in the MG. Thus to retain information from the previous setting the MGC must include that information in

the new setting. If the MGC wishes to delete some information from the existing descriptor, it merely resends the descriptor (in a Modify command) with the unwanted information stripped out.

7.1.9. Events Descriptor

The EventsDescriptor parameter contains a RequestIdentifier and a list of events that the Media Gateway is requested to detect and report. The RequestIdentifier is used to correlate the request with the notifications that it may trigger. Requested events include, for example, fax tones, continuity tones, and on-hook and off-hook transitions.

Each event in the descriptor contains the Event name, an optional streamID, an optional KeepActive flag, and optional parameters. The Event name consists of a Package Name (where the event is defined) and an EventID. The ALL wildcard may be used for the EventID, indicating that all events from the specified package have to be detected. The default streamIDis 0, indicating that the event to be detected is not related to a particular media stream. Events can have parameters. This

allows a single event description to have some variation in meaning without creating large numbers of individual events. Further event parameters are defined in the package.

The MG shall send a Notify command to the MGC when it detects an event in the Events Descriptor. If the EventBuffer flag is "on", following detection of such an event, normal handling of events is suspended, and any event found in the EventBuffer Descriptor which is subsequently detected is added to the end of a FIFO queue, along with the time that it was detected. A command containing an Events Descriptor which is received when the EventBuffer flag is on causes the following sequence to be executed:

1. The first event in the FIFO queue is examined. If it is in the Events listed in the new events descriptor, the MG shall send a Notify command to the MGC and remove the event from the FIFO queue. The time stamp of the Notify shall be the time the event was actually detected.

2. If the event is not in the new Events Descriptor, it shall be discarded.

3. If the queue is empty, the sequence shall be stopped, and normal event processing shall be resumed. If there are any events remaining in the queue, the sequence repeats.

If the EventBuffer flag is off when the new Events Descriptor is received, the queue is flushed, and no events are added to it. The default state of EventBuffer is off.

Normally, detection of an event shall cause any active signals to stop. When KeepActive is specified in the event, the MG shall not interrupt any signals active on the Termination on which the event is detected.

An event can include an Embedded Signals descriptor and/or an Embedded Events Descriptor which, if present, replaces the current Signals/Events descriptor when the event is detected. It is possible, for example, to specify that the dial-tone Signal be generated when an off-hook Event is detected, or that the dial-tone Signal be stopped when a digit is detected. A media gateway controller shall not send EventsDescriptors with an event both marked KeepActive and containing an embedded SignalsDescriptor.

Only one level of embedding is permitted. An embedded EventsDescriptor SHALL NOT contain another embedded EventsDescriptor.

An Events Descriptor received by a media gateway replaces any previous Events Descriptor. Event notification in process shall complete, and events detected after the command containing the new EventsDescriptor executes, shall be processed according to the new EventsDescriptor.

7.1.10. EventBuffer Descriptor

The EventBuffer Descriptor contains a list of events, with their parameters if any, that the MG is requested to detect and buffer when no Events Descriptor is active (See 7.1.9).

7.1.11. Signals Descriptor

A SignalsDescriptor is a parameter that contains the set of signals that the Media Gateway is asked to apply to a Termination. A SignalsDescriptor contains a number of signals and/or sequential signal lists. A SignalsDescriptor may contain zero signals and sequential signal lists. Support of sequential signal lists is optional.

Signals are defined in packages. Signals shall be named with a Package name (in which the signal is defined) and a SignalID. No wildcard shall be used in the SignalID. Signals that occur in a SignalsDescriptor have an optional StreamID parameter (default is 0, to indicate that the signal is not related to a particular media stream), an optional signal type (see below), an optional duration and possibly parameters defined in the package that defines the signal. This allows a single signal to have some variation in meaning, obviating the need to create large numbers of individual signals. Finally, the optional parameter "notifyCompletion" allows the MGC to indicate that it wishes to be notified when this signal finishes playout. When the MGC enables the Signal Completion event (see section E.1.2) in an Event Descriptor, that event is detected whenever a signal terminates and "notifyCompletion" for that

signal is set to TRUE..

The duration is an integer value that is expressed in hundredths of a second.

There are three types of signals:

* on/off - the signal lasts until it is turned off,

* timeout - the signal lasts until it is turned off or a specific period of time elapses,

* brief - the signal duration is so short that it will stop on its own unless a new signal is applied that causes it to stop; no timeout value is needed.

If the signal type is specified in a SignalsDescriptor, it overrides the default signal type (see Section 12.1.4). If duration is specified for an on/off signal, it SHALL be ignored.

A sequential signal list consists of a signal list identifier, a sequence of signals to be played sequentially, and a signal type. Only the trailing element of the sequence of signals in a sequential signal list may be an on/off signal. If the trailing element of the sequence is an on/off signal, the signal type of the sequential signal list shall

be on/off as well. If the sequence of signals in a sequential signal list contains signals of type timeout and the trailing element is not of type on/off, the type of the sequential signal list SHALL be set to timeout. The duration of a sequential signal list with type timeout is the sum of the durations of the signals it contains. If the sequence of signals in a sequential signal list contains only signals of type brief, the type of the sequential signal list SHALL be set to brief. A signal list is treated as a single signal of the specified type when played out.

Multiple signals and sequential signal lists in the same SignalsDescriptor shall be played simultaneously.

Signals are defined as proceeding from the termination towards the exterior of the Context unless otherwise specified in a package. When the same Signal is applied to multiple Terminations within one Transaction, the MG should consider using the same resource to generate these Signals.

Production of a Signal on a Termination is stopped by application of a new SignalsDescriptor, or detection of an Event on the Termination (see section 7.1.9).

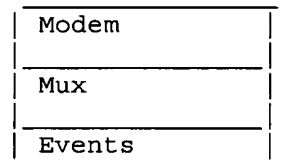
A new SignalsDescriptor replaces any existing SignalsDescriptor. Any signals applied to the Termination not in the replacement descriptor shall be stopped, and new signals are applied. Signals present in both the existing and replacement descriptor, with the same parameters in both, shall be continued. If the replacement descriptor contains a sequential signal list with the same identifier as the existing descriptor, then

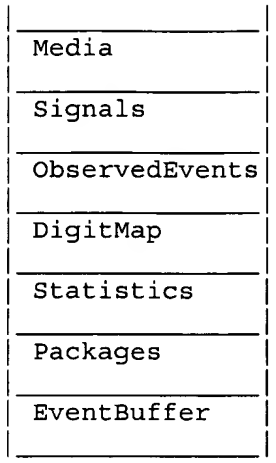
* the signal type and sequence of signals in the sequential signal list in the replacement descriptor shall be ignored, and

* the playing of the signals in the sequential signal list in the existing descriptor shall not be interrupted.

7.1.12. Audit Descriptor

Specifies what information is to be audited. The Audit Descriptor specifies the list of descriptors to be returned. Audit may be used in any command to force the return of a descriptor even if the descriptor in the command was not present, or had no underspecified parameters. Possible items in the Audit Descriptor are:





Audit may be empty, in which case, no descriptors are returned. This is useful in Subtract, to inhibit return of statistics, especially when using wildcard.

7.1.13. ServiceChange Descriptor

The ServiceChangeDescriptor contains the following parameters:

* ServiceChangeMethod
* ServiceChangeReason
* ServiceChangeAddress
* ServiceChangeDelay
* ServiceChangeProfile
* ServiceChangeVersion
* ServiceChangeMGCId
* TimeStamp

See section 7.2.8

7.1.14. DigitMap Descriptor

A DigitMap is a dialing plan resident in the Media Gateway used for detecting and reporting digit events received on a Termination. The DigitMap Descriptor contains a DigitMap name and the DigitMap to be assigned. A digit map may be preloaded into the MG by management action and referenced by name in an EventsDescriptor, may be defined dynamically and subsequently referenced by name, or the actual digitmap itself may be specified in the EventsDescriptor.

DigitMaps defined in a DigitMapDescriptor can occur in any of the standard Termination manipulation Commands of the protocol. A DigitMap, once defined, can be used on all Terminations specified by the (possibly wildcarded) TerminationID in such a command. When a DigitMap is defined dynamically in a DigitMap Descriptor:

* A new DigitMap is created by specifying a name that is not yet defined. The value shall be present.

* A DigitMap value is updated by supplying a new value for a name that is already defined. Terminations presently using the digitmap shall continue to use the old definition; subsequent EventsDescriptors specifying the name, including any EventsDescriptor in the command containing the DigitMap descriptor, shall use the new one.

* A DigitMap is deleted by supplying an empty value for a name that

is already defined. Terminations presently using the digitmap shall continue to use the old definition.

The collection of digits according to a DigitMap may be protected by three timers, viz. a start timer (T), short timer (S), and long timer (L).

1. The start timer (T) is used prior to any digits having been dialed.

2. If the Media Gateway can determine that at least one more digit is needed for a digit string to match any of the allowed patterns in the digit map, then the interdigit timer value should be set to a long (L) duration (e.g.-16 seconds).

3. If the digit string has matched one of the patterns in the digit map, but it
is possible that more digits could be received which would cause a match
with a different pattern, then instead of reporting the match immediately,
the MG must apply the short timer (S) and wait for more digits.

The timers are configurable parameters to a DigitMap. The Start timer is started at the beginning of every digit map use, but can be overridden.

The formal syntax of the digit map is described by the DigitMap rule in the formal syntax description of the protocol (see Annex A and Annex B). A DigitMap, according to this syntax, is defined either by a string or by a list of strings. Each string in the list is an alternative event sequence, specified either as a sequence of digit map symbols or as a regular expression of digit map symbols. These digit map symbols, the digits "0" through "9" and letters "A" through a maximum value depending on the signalling system concerned, but never exceeding "K", correspond

to specified events within a package which has been designated in the Events Descriptor on the termination to which the digit map is being applied. (The mapping between events and digit map symbols is defined in the documentation for packages associated with channel-associated signalling systems such as DTMF, MF, or R2. Digits "0" through "9" MUST be mapped to the corresponding digit events within the signalling system concerned. Letters should be allocated in logical fashion, facilitating the use of range notation for alternative events.) The letter "x" is used as a wildcard, designating any event corresponding to symbols in the range "0"-"9". The string may also contain explicit ranges and, more generally, explicit sets of symbols, designating alternative events any one of which satisfies that position of the digit map. Finally, the dot symbol "." stands for zero or more repetitions of the event selector (event, range of events, set of alternative events, or wildcard) which

precedes it. As a consequence of the third timing rule above, inter-event timing while matching the dot symbol uses the short timer.

In addition to these event symbols, the string may contain "L" duration modifiers. An "L" designates a long event: placed in front of the symbol(s) designating the event(s) which satisfy a given digit position, it indicates that that position is satisfied only if the duration of the event exceeds the long-duration threshold. The value of this threshold is assumed to be provisioned in the MG.

A digit map is active while the event descriptor which invoked it is active and it has not completed. A digit map completes when:

* a timer has expired, or

* an alternative event sequence has been matched and no other alternative event sequence in the digit map could be matched through detection of an additional event (unambiguous match), or

* an event has been detected such that a match to a complete alternative event sequence of the digit map will be impossible no matter what additional events are received.

Upon completion, a digit map completion event as defined in the package providing the events being mapped into the digit map shall be generated. Pending completion, successive events shall be processed according to the following rules:

The "current dial string", an internal variable, is initially empty. The set of candidate alternative event sequences includes all of the alternatives specified in the digit map.

At each step, a timer is set to wait for the next event. The rules for determining how long to wait are listed above. If the timer expires and a member of the candidate set of alternatives is fully satisfied, a timeout completion with full match is reported. If the timer expires and part or none of any candidate alternative is satisfied, a timeout com-

pletion with partial match is reported.

If an event is detected before the timer expires, it is mapped to a igit string symbol and added to the end of the current dial string. The duration of the event (long or not long) is noted if and only if this is relevant in the current symbol position (because at least one of the candidate alternative event sequences includes the "L" modifier at this position in the sequence).

The current dial string is compared to the candidate alternative event sequences, and any of these which do not match are discarded from the

candidate set. If a sequence expecting a long-duration event at this position is matched (i.e. the event had long duration and met the specification for this position), then any alternative event sequences not specifying a long duration event at this position are discarded, and the current dial string is modified by inserting an "L" in front of the symbol representing the latest event. If no sequence expecting a long-duration event at this position is matched, any such sequences are discarded from the candidate set, but the observed event duration is treated as irrelevant in assessing matches to the remaining candidates.

If exactly one candidate remains, a completion event is generated indicating an unambiguous match. If no candidates remain, but one of the candidates from the previous step was fully satisfied before the latest event was detected, a completion event is generated indicating a full match and an extra event. If no candidates remain and no candidate from the previous step was fully satisfied before the latest event was detected, a completion event is generated indicating a partial match and an extra event. If multiple candidates remain, steps 1 through 4 are repeated.

As an example, consider the following dial plan:

0	Local operator
00	Long distance operator
xxxx	Local extension number
8xxxxxxx	Local number
#xxxxxxx	Off-site extension
*xx	Star services
91xxxxxxxxxx	Long distance number
9011 + up to 15 digits	International number

If the DTMF detection package described in Annex E (section E.6) is used to collect the dialed digits, then the dialing plan shown above results in the following digit map:

```
(0 | 00 | [1-7]xxx | 8xxxxxxx | Fxxxxxxx | Exx | 91xxxxxxxxxx | 9011x. )
```

7.1.15. Statistics Descriptor

The Statistics parameter provides information describing the status and usage of a Termination during its existence within a specific Context. There is a set of standard statistics kept for each termination where appropriate (number of octets sent and received for example). The particular statistical properties that are reported for a given Termination

are determined by the Packages realized by the Termination. By default, statistics are reported when the Termination is Subtracted from the Context. This behavior can be overridden by including an empty AuditDescriptor in the Subtract command. Statistics may also be returned from the AuditValue command, or any Add/Move/Modify command using the Audit descriptor.

Statistics are cumulative; reporting Statistics does not reset them. Statistics are reset when a Termination is Subtracted from a Context.

7.1.16. Packages Descriptor

Used only with the AuditValue command, the PackageDescriptor returns a list of Packages realized by the Termination.

7.1.17. ObservedEvents Descriptor

ObservedEvents is supplied with the Notify command to inform the MGC of which event(s) were detected. Used with the AuditValue command, the ObservedEventsDescriptor returns events in the event buffer which have not been Notified. In addition, if a digit map is active, the ObservedEventsDescriptor shall contain a digit map completion event as defined in the package whose events are being mapped. This event shall show the contents of the current dial string at the time the audit request was processed. ObservedEvents contains the RequestIdentifier of the EventsDescriptor that triggered the notification, the event(s) detected and the detection time(s). Detection times are reported with a precision of hundredths of a second. Time is expressed in UTC.

7.1.18. Topology Descriptor

A topology descriptor is used specify flow directions between terminations in a Context. Contrary to the descriptors in previous sections, the topology descriptor applies to a Context instead of a Termination. The default topology of a Context is that each termination's transmission is received by all other terminations. The Topology Descriptor is optional to implement.

The Topology Descriptor occurs before the commands in an action. It is possible to have an action containing only a Topology Descriptor, provided that the context to which the action applies already exists.

A topology descriptor consists of a sequence of triples of the form (T1,

T2, association). T1 and T2 specify Terminations within the Context, possibly using the ALL or CHOOSE wildcard. The association specifies how media flows between these two Terminations as follows.

* (T1, T2, isolate) means that the Terminations matching T2 do not

receive media from the Terminations matching T1, nor vice versa.

* (T1, T2, oneway) means that the Terminations that match T2 receive media from the Terminations matching T1, but not vice versa. In this case use of the ALL wildcard such that there are Terminations that match both T1 and T2 is not allowed.

* (T1, T2, bothway) means that the Terminations matching T2 receive media from the Terminations matching T1, and vice versa. In this case it is allowed to use wildcards such that there are Terminations that match both T1 and T2. However, if there is a Termination that matches both, no loopback is introduced; loopbacks are created by setting the TerminationMode.

CHOOSE wildcards may be used in T1 and T2 as well, under the following restrictions:

* the action (see section 8) of which the topology descriptor is part contains an Add command in which a CHOOSE wildcard is used;

* if a CHOOSE wildcard occurs in T1 or T2, then a partial name SHALL NOT be specified.

The CHOOSE wildcard in a topology descriptor matches the TerminationID that the MG assigns in the first Add command that uses a CHOOSE wildcard in the same action. An existing Termination that matches T1 or T2 in the Context to which a Termination is added, is connected to the newly added Termination as specified by the topology descriptor. The default association when a termination is not mentioned in the Topology descriptor is bothway (if T3 is added to a context with T1 and T2 with topology (T3,T1,oneway) it will be connected bothway to T2).

The figure below and the table following it show some examples of the effect of including topology descriptors in actions.

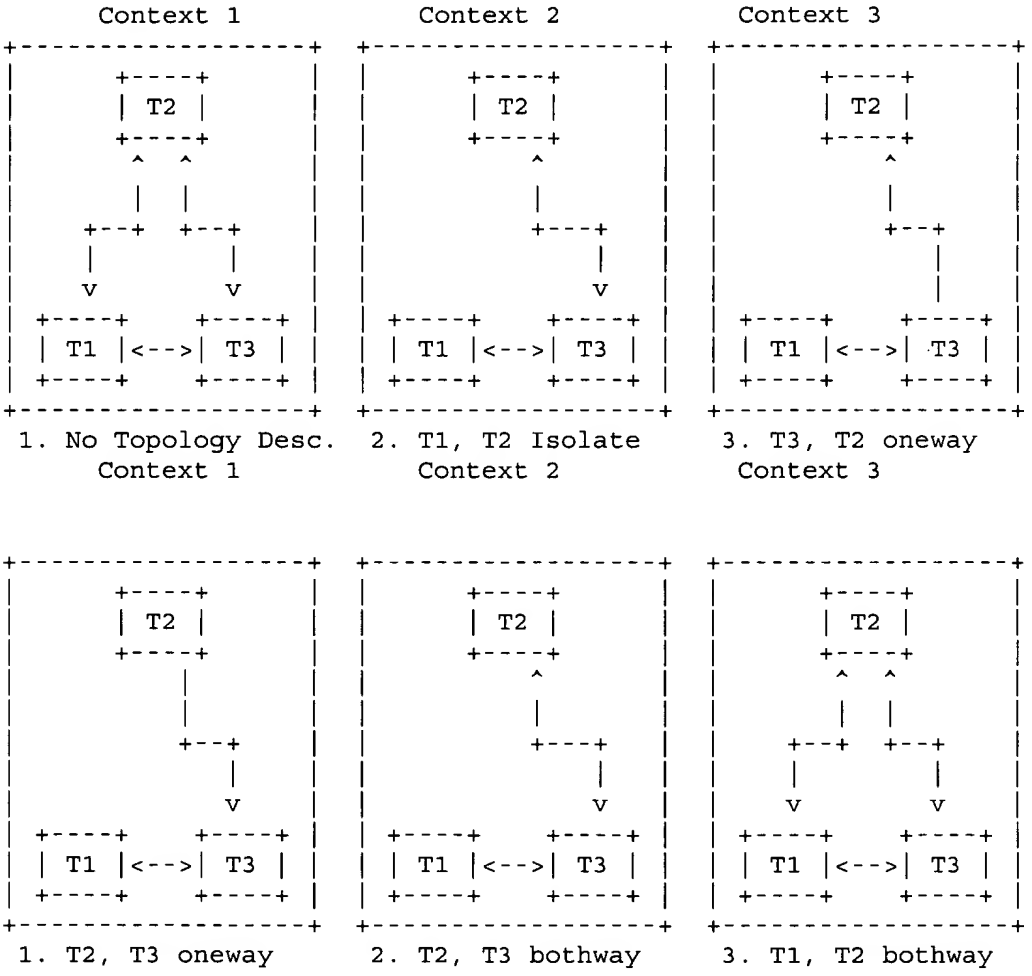


Figure 4: Example topologies

Topology	Description
1	No topology descriptors. When no topology descriptors are included, all terminations have a both way connection to all other terminations.
2	T1, T2, Isolated. Removes the connection between T1 and T2. T3 has a both way connection with both T1 and T2.
3	T3, T2, oneway. A oneway connection from T3 to T2 (i.e. T2 receives media flow from T3). A bothway connection between T1 and T3.
4	T2, T3, oneway. A oneway connection between T2 to T3. T1 and T3 remain bothway connected
5	T2, T3 bothway. T2 is bothway connected to T3. This results in the same as 2.
6	T1, T2 bothway. (T2, T3 bothway and T1,T3 bothway may be implied or explicit). terminations have a bothway

A oneway connection must implemented in such a way that the other Terminations in the Context are not aware of the change in topology.

7.2. Command Application Programming Interface

Following is an Application Programming Interface (API) describing the Commands of the protocol. This API is shown to illustrate the Commands and their parameters and is not intended to specify implementation (e.g. via use of blocking function calls). It describes the input parameters in parentheses after the command name and the return values in front of the Command. This is only for descriptive purposes; the actual Command syntax and encoding are specified in later subsections. All parameters enclosed by square brackets ([. . .]) are considered optional.

7.2.1. Add

The Add Command adds a Termination to a Context.

```
TerminationID
[,MediaDescriptor]
[,ModemDescriptor]
[,MuxDescriptor]
```

```
[,EventsDescriptor]
[,SignalsDescriptor]
[,DigitMapDescriptor]
[,ObservedEventsDescriptor]
[,EventBufferDescriptor]
[,StatisticsDescriptor]
[,PackagesDescriptor]
     Add( TerminationID
          [, MediaDescriptor]
           [, ModemDescriptor]
           [, MuxDescriptor]
           [, EventsDescriptor]
           [, SignalsDescriptor]
           [, DigitMapDescriptor]
           [, AuditDescriptor]
     )
```

The TerminationID specifies the termination to be added to the Context. The Termination is either created, or taken from the null Context. For an existing Termination, the TerminationID would be specific. For a Termination that does not yet exist, the TerminationID is specified as CHOOSE in the command. The new TerminationID will be returned. Wild-cards may be used in an Add, but such usage would be unusual. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified.

The optional MediaDescriptor describes all media streams.

The optional ModemDescriptor and MuxDescriptor specify a modem and multiplexer if applicable. For convenience, if a Multiplex Descriptor is present in an Add command and lists any Terminations that are not currently in the Context, such Terminations are added to the context as if individual Add commands listing the Terminations were invoked. If an error occurs on such an implied Add, error 471 - Implied Add for Multiplex failure shall be returned and further processing of the command shall cease.

The EventsDescriptor parameter is optional. If present, it provides the list of events that should be detected on the Termination.

The SignalsDescriptor parameter is optional. If present, it provides the list of signals that should be applied to the Termination.

The DigitMapDescriptor parameter is optional. If present, defines a DigitMap definition that may be used in an EventsDescriptor.

The AuditDescriptor is optional. If present, the command will return descriptors as specified in the AuditDescriptor.

All descriptors that can be modified could be returned by MG if a parameter was underspecified or overspecified. ObservedEvents, Statistics, and Packages, and the EventBuffer Descriptors are returned only if requested in the AuditDescriptor. Add SHALL NOT be used on a Termination with a serviceState of "OutofService".

7.2.2. Modify

The Modify Command modifies the properties of a Termination.

```
TerminationID
[,MediaDescriptor]
[,ModemDescriptor]
[,MuxDescriptor]
[,EventsDescriptor]
[,SignalsDescriptor]
[,DigitMapDescriptor]
[,ObservedEventsDescriptor]
[,EventBufferDescriptor]
[,StatisticsDescriptor]
[,PackagesDescriptor]
     Modify( TerminationID
             [, MediaDescriptor]
             [, ModemDescriptor]
             [, MuxDescriptor]
             [, EventsDescriptor]
             [, SignalsDescriptor]
             [, DigitMapDescriptor]
             [, AuditDescriptor]
     )
```

The TerminationID may be specific if a single Termination in the Context is to be modified. Use of wildcards in the TerminationID may be appropriate for some operations. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified. The CHOOSE option is an error, as the Modify command may only be used on existing Terminations.

The remaining parameters to Modify are the same as those to Add. Possible return values are the same as those to Add. Modify SHALL NOT be used on a Termination with a serviceState of "OutofService".

7.2.3. Subtract

The Subtract Command disconnects a Termination from its Context and returns statistics on the Termination's participation in the Context.

```
TerminationID
[,MediaDescriptor]
[,ModemDescriptor]
[,MuxDescriptor]
[,EventsDescriptor]
[,SignalsDescriptor]
[,DigitMapDescriptor]
[,ObservedEventsDescriptor]
[,EventBufferDescriptor]
[,StatisticsDescriptor]
[,PackagesDescriptor]
     Subtract(TerminationID
              [, AuditDescriptor]
     )
```

TerminationID in the input parameters represents the Termination that is being subtracted. The TerminationID may be specific or may be a wild-card value indicating that all (or a set of related) Terminations in the Context of the Subtract Command are to be subtracted. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified. The CHOOSE option is an error, as the Subtract command may only be used on existing Terminations. ALL may be used as the ContextID as well as the TerminationId in a Subtract, which would have the effect of deleting all contexts, deleting all ephemeral terminations, and returning all physical terminations to Null context.

By default, the Statistics parameter is returned to report information collected on the Termination or Terminations specified in the Command. The information reported applies to the Termination's or Terminations'Termination's or Terminations' existence in the Context from which it or they are being subtracted.

The AuditDescriptor is optional. If present, the command will return descriptors as specified in the AuditDescriptor. Possible return values are the same as those to Add.

When a provisioned Termination is Subtracted from a context, its property values shall revert to:

* The default value, if specified for the property and not overridden by provisioning or modification within the null context

* The provisioned value, if not overridden by modification in the null context

* The last value set by a modification while the termination was in the null context.

7.2.4. Move

The Move Command moves a Termination to another Context from its current Context in one atomic operation. The Move command is the only command that refers to a Termination in a Context different from that to which the command is applied. The Move command shall not be used to move Terminations to or from the null Context.

```
TerminationID
[,MediaDescriptor]
[,ModemDescriptor]
[,MuxDescriptor]
[,EventsDescriptor]
[,SignalsDescriptor]
[,DigitMapDescriptor]
[,ObservedEventsDescriptor]
[,EventBufferDescriptor]
[,StatisticsDescriptor]
[,PackagesDescriptor]
     Move( TerminationID
           [, MediaDescriptor]
           [, ModemDescriptor]
           [, MuxDescriptor]
           [, EventsDescriptor]
           [, SignalsDescriptor]
           [, DigitMapDescriptor]
           [, AuditDescriptor]
     )
```

The TerminationID specifies the Termination to be moved. It may be wildcarded. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified. By convention, the Termination is subtracted from its previous Context. The Context to which the Termination is moved is indicated by the target ContextId in the Action. If the last remaining Termination is moved out of a Context, the Context is deleted.

The remaining descriptors are processed as in the Modify Command. The AuditDescriptor with the Statistics option, for example, would return statistics on the Termination just prior to the Move. Possible descrip-

tors returned from Move are the same as for Add. Move SHALL NOT be used on a Termination with a serviceState of "OutofService".

7.2.5. AuditValue

The AuditValue Command returns the current values of properties, events, signals and statistics associated with Terminations.

```
TerminationID
[,MediaDescriptor]
[,ModemDescriptor]
[,MuxDescriptor]
[,EventsDescriptor]
[,SignalsDescriptor]
[,DigitMapDescriptor]
[,ObservedEventsDescriptor]
[,EventBufferDescriptor]
[,StatisticsDescriptor]
[,PackagesDescriptor]
        AuditValue(TerminationID,
                AuditDescriptor
        )
```

TerminationID may be specific or wildcarded. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified. If a wildcarded response is requested, only one command return is generated, with the contents containing the union of the values of all Terminations matching the wildcard. This convention may reduce the volume of data required to audit a group of Terminations. Use of CHOOSE is an error.

The appropriate descriptors, with the current values for the Termination, are returned from AuditValue. Values appearing in multiple instances of a descriptor are defined to be alternate values supported, with each parameter in a descriptor considered independent.

ObservedEvents returns a list of events in the EventBuffer, PackagesDescriptor returns a list of packages realized by the Termination. DigitMapDescriptor returns the name or value of the current DigitMap for the Termination. DigitMap applied to the root Termination returns all named DigitMaps in the gateway. Statistics returns the current values of all statistics being kept on the Termination. Specifying an empty

Audit Descriptor results in only the TerminationID being returned. This may be useful to get a list of TerminationIDs when used with wildcard.

AuditValue results depend on the Context, viz. specific, null, or wildcarded. The TerminationID may be specific, or wildcarded.

The following illustrates other information that can be obtained with the Audit Command:

ContextID	TerminationID	Information Obtained
Specific	wildcard	Audit of matching Terminations in a Context
Specific	specific	Audit of a single Termination in a Context
Null	Root	Audit of Media Gateway state and events
Null	wildcard	Audit of all matching Terminations
Null	specific	Audit of a single Termination outside of any Context
All	wildcard	Audit of all matching Terminations and the Context to which they are associated
All	Root	List of all ContextIds

7.2.6. AuditCapabilities

The AuditCapabilities Command returns the possible values of properties, events, signals and statistics associated with Terminations.

```
TerminationID
[,MediaDescriptor]
[,ModemDescriptor]
[,MuxDescriptor]
[,EventsDescriptor]
[,SignalsDescriptor]
[,ObservedEventsDescriptor]
[,EventBufferDescriptor]
[,StatisticsDescriptor]
        AuditCapabilities(TerminationID,
                AuditDescriptor)
```

The appropriate descriptors, with the possible values for the Termination are returned from AuditCapabilities. Descriptors may be repeated where there are multiple possible values. values. If a wildcarded response is requested, only one command return is generated, with the contents containing the union of the values of all Terminations matching the wildcard. This convention may reduce the volume of data required to

audit a group of Terminations.

Interpretation of what capabilities are requested for various values of ContextID and TerminationID is the same as in AuditValue.

The EventsDescriptor returns the list of possible events on the Termination together with the list of all possible values for the EventsDescriptor Parameters. The SignalsDescriptor returns the list of possible signals that could be applied to the Termination together with

the list of all possible values for the Signals Parameters. StatisticsDescriptor returns the names of the statistics being kept on the termination. ObservedEventsDescriptor returns the names of active events on the termination. DigitMap and Packages are not legal in AuditCapability

7.2.7. Notify

The Notify Command allows the Media Gateway to notify the Media Gateway Controller of events occurring within the Media Gateway.

```
Notify(TerminationID,
          ObservedEventsDescriptor)
```

The TerminationID parameter specifies the Termination issuing the Notify Command. The TerminationID shall be a fully qualified name.

The ObservedEventsDescriptor contains the RequestID and a list of events that the Media Gateway detected in the order that they were detected. Each event in the list is accompanied by parameters associated with the event and an indication of the time that the event was detected. Notify Commands shall occur only as the result of detection of an event specified by an Events Descriptor which is active on the termination concerned.

The RequestID returns the RequestID parameter of the EventsDescriptor that triggered the Notify Command. It is used to correlate the notification with the request that triggered it. The events in the list must have been requested via the triggering EventsDescriptor or embedded EventsDescriptor.

7.2.8. ServiceChange

The ServiceChange Command allows the Media Gateway to notify the Media Gateway Controller that a Termination or group of Terminations is about to be taken out of service or has just been returned to service. The Media Gateway Controller may indicate that Termination(s) shall be taken out of or returned to service. The Media Gateway may notify the MGC

that the capability of a Termination has changed. It also allows a MGC to hand over control of a MG to another MGC.

```
TerminationID
[ServiceChangeReplyDescriptor]
        ServiceChange(TerminationID,
                ServiceChangeDescriptor
        )
```

The TerminationID parameter specifies the Termination(s) that are taken out of or returned to service. Wildcarding of Termination names is per-

mitted, with the exception that the CHOOSE mechanism shall not be used. Use of the "Root" TerminationID indicates a ServiceChange affecting the entire Media Gateway.

The ServiceChangeDescriptor contains the following parameters as required:

* ServiceChangeMethod

* ServiceChangeReason

* ServiceChangeDelay

* ServiceChangeAddress

* ServiceChangeProfile

* ServiceChangeVersion

* ServiceChangeMGCId

* TimeStamp

The ServiceChangeMethod parameter specifies the type of ServiceChange that will or has occurred:

1) Graceful - indicates that the specified Terminations will be taken out of service after the specified ServiceChangeDelay; established connections are not yet affected, but the Media Gateway Controller should refrain from establishing new connections and should attempt to gracefully tear down existing connections. The MG should set termination serviceState to "test" until the expiry of ServiceChangeDelay or the removal of the termination from an active context (whichever is first), then set it to "out of service".

2) Forced - indicates that the specified Terminations were taken

abruptly out of service and any established connections associated with them were lost. The MGC is responsible for cleaning up the context (if any) with which the failed termination is associated. At a minimum the termination shall be subtracted from the context. The termination serviceState should be "out of service".

3) Restart - indicates that service will be restored on the specified Terminations after expiration of the ServiceChangeDelay. The serviceState should be set to "inService" upon expiry of ServiceChangeDelay.

4) Disconnected - always applied with the Root TerminationID, indicates that the MG lost communication with the MGC, but it was subsequently restored. Since MG state may have changed, the MGC may wish to use the Audit command to resynchronize its state with the

MG's.

5) Handoff - sent from the MGC to the MG, this reason indicates that the MGC is going out of service and a new MGC association must be established. Sent from the MG to the MGC, this indicates that the MG is attempting to establish a new association in accordance with a Handoff received from the MGC with which it was previously associated.

6) Failover - sent from MG to MGC to indicate the primary MG is out of service and a secondary MG is taking over, and sent from MG to (new) MGC in response to the MG having received a ServiceChange with ServiceChangeMethod equal to Handoff.

7) Another value whose meaning is mutually understood between the MG and the MGC.

The ServiceChangeReason parameter specifies the reason why the ServiceChange has or will occur. It consists of an alphanumeric token (IANA registered) and an explanatory string.

The optional ServiceChangeAddress parameter specifies the address (e.g., IP port number for IP networks) to be used for subsequent communications. It can be specified in the input parameter descriptor or the returned result descriptor. ServiceChangeAddress and ServiceChangeMgcId parameters shall not both be present in the ServiceChangeDescriptor or the ServiceChangeResultDescriptor. The ServiceChangeAddress provides an address to be used within the context of the association currently being negotiated, while the ServiceChangeMgcId provides an alternate address where the MG should seek to establish another association.

The optional ServiceChangeDelay parameter is expressed in seconds. If the delay is absent or set to zero, the delay value should be considered

to be null. In the case of a "graceful" ServiceChangeMethod, a null delay indicates that the Media Gateway Controller should wait for the natural removal of existing connections and should not establish new connections. . For "graceful" only, null delay means the MG should set serviceState to "test" immediately, then wait indefinitely for the termination to be removed from any active context before setting serviceState to "out of service". For "restart", null means immediate return to service.

The optional ServiceChangeProfile parameter specifies the Profile (if any) of the protocol supported. The ServiceChangeProfile includes the version of the profile supported.

The optional ServiceChangeVersion parameter contains the protocol version and is used if protocol version negotiation occurs (see section 11.3).

The optional TimeStamp parameter specifies the actual time as kept by

the sender. It can be used by the responder to determine how its notion of time differs from that of its correspondent. TimeStamp is sent with a precision of hundredths of a second, and is expressed in UTC.

The optional Extension parameter may contain any value whose meaning is mutually understood by the MG and MGC.

A ServiceChange Command specifying the "Root" for the TerminationID and ServiceChangeMethod equal to Restart is a registration command by which a Media Gateway announces its existence to the Media Gateway Controller. The Media Gateway is expected to be provisioned with the name of one primary and optionally some number of alternate Media Gateway Controllers.

Acknowledgement of the ServiceChange Command completes the registration process. The MG may specify an address in the ServiceChangeAddress parameter of the ServiceChange request, and the MGC may also do so in the ServiceChange reply. In either case, the recipient must use the supplied address as the destination for all subsequent transaction requests within the association. At the same time, as indicated in section 9, transaction replies and pending indications must be sent to the address from which the corresponding requests originated. This must be done even if it implies extra messaging because commands and responses cannot be packed together. The TimeStamp parameter shall be sent with a registration command and its response.

The Media Gateway Controller may return a ServiceChangeMGCId parameter that describes the Media Gateway Controller that should preferably be contacted for further service by the Media Gateway. In this case the Media Gateway shall reissue the ServiceChange command to the new Media Gateway Controller. The Gateway specified in a ServiceChangeMGCId, if

provided, shall be contacted before any further alternate MGCs. On a HandOff message from MGC to MG, the ServiceChangeMGCId is the new MGC that will take over from the current MGC.

The return from ServiceChange is empty except when the Root terminationID is used. In that case it includes the following parameters as required:

* ServiceChangeAddress, if the responding MGC wishes to specify an new destination for messages from the MG for the remainder of the association;

* ServiceChangeMgcId, if the responding MGC does not wish to sustain an association with the MG;

* ServiceChangeProfile, if the responder wishes to negotiate the profile to be used for the association;

* ServiceChangeVersion, if the responder wishes to negotiate the version of the protocol to be used for the association.

The following ServiceChangeReasons are defined. This list may be extended by an IANA registration as outlined in section 13.3

900 Service Restored
901 MG Cold Boot
902 MG Warm Boot
903 MGC Directed Change
904 Termination malfunctioning
905 Termination taken out of service
906 Loss of lower layer connectivity (e.g. downstream sync)
907 Transmission Failure
908 MG Impending Failure
909 MGC Impending Failure
910 Media Capability Failure
911 Modem Capability Failure
912 Mux Capability Failure
913 Signal Capability Failure
914 Event Capability Failure
915 State Loss

7.2.9. Manipulating and Auditing Context Attributes

The commands of the protocol as discussed in the preceding sections apply to terminations. This section specifies how contexts are manipulated and audited.

Commands are grouped into actions (see section 8). An action applies to one context. In addition to commands, it may contain context manipulation and auditing instructions.

An action request sent to a MG may include a request to audit attributes of a context. An action may also include a request to change the attributes of a context.

The context properties that may be included in an action reply are used to return information to a MGC. This can be information requested by an audit of context attributes or details of the effect of manipulation of a context.

If a MG receives an action which contains both a request to audit context attributes and a request to manipulate those attributes, the response SHALL include the values of the attributes after processing the manipulation request.

7.2.10. Generic Command Syntax

The protocol can be encoded in a binary format or in a text format. MGCs should support both encoding formats. MGs may support both formats.

The protocol syntax for the binary format of the protocol is defined in Annex A. Annex C specifies the encoding of the Local and Remote descriptors for use with the binary format.

A complete ABNF of the text encoding of the protocol per RFC2234 is given in Annex B. SDP, as modified herein is used as the encoding of the Local and Remote Descriptors for use with the text encoding.

7.3. Command Error Codes

Errors consist of an IANA registered error code and an explanatory string. Sending the explanatory string is optional. Implementations are encouraged to append diagnostic information to the end of the string.

When a MG reports an error to a MGC, it does so in an error descriptor. An error descriptor consists of an error code and optionally the associated explanatory string.

The identified error codes are:

400 - Bad Request
401 - Protocol Error
402 - Unauthorized

403 - Syntax Error in Transaction
404 - Syntax Error in TransactionReply
405 - Syntax Error in TransactionPending
406 - Version Not Supported
410 - Incorrect identifier
411 - The transaction refers to an unknown ContextId
412 - No ContextIDs available
421 - Unknown action or illegal combination of actions
422 - Syntax Error in Action
430 - Unknown TerminationID
431 - No TerminationID matched a wildcard
432 - Out of TerminationIDs or No TerminationID available
433 - TerminationID is already in a Context
440 - Unsupported or unknown Package
441 - Missing RemoteDescriptor
442 - Syntax Error in Command
443 - Unsupported or Unknown Command
444 - Unsupported or Unknown Descriptor
445 - Unsupported or Unknown Property
446 - Unsupported or Unknown Parameter
447 - Descriptor not legal in this command
448 - Descriptor appears twice in a command
450 - No such property in this package
451 - No such event in this package
452 - No such signal in this package
453 - No such statistic in this package
454 - No such parameter value in this package

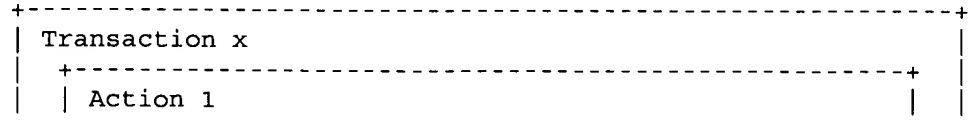
455 - Parameter illegal in this Descriptor
456 - Parameter or Property appears twice in this Descriptor
461 - TransactionIDs in Reply do not match Request
462 - Commands in Transaction Reply do not match commands in request
463 - TerminationID of Transaction Reply does not match request
464 - Missing reply in Transaction Reply
465 - TransactionID in Transaction Pending does not match any open request
466 - Illegal Duplicate Transaction Request
467 - Illegal Duplicate Transaction Reply
471 - Implied Add for Multiplex failure

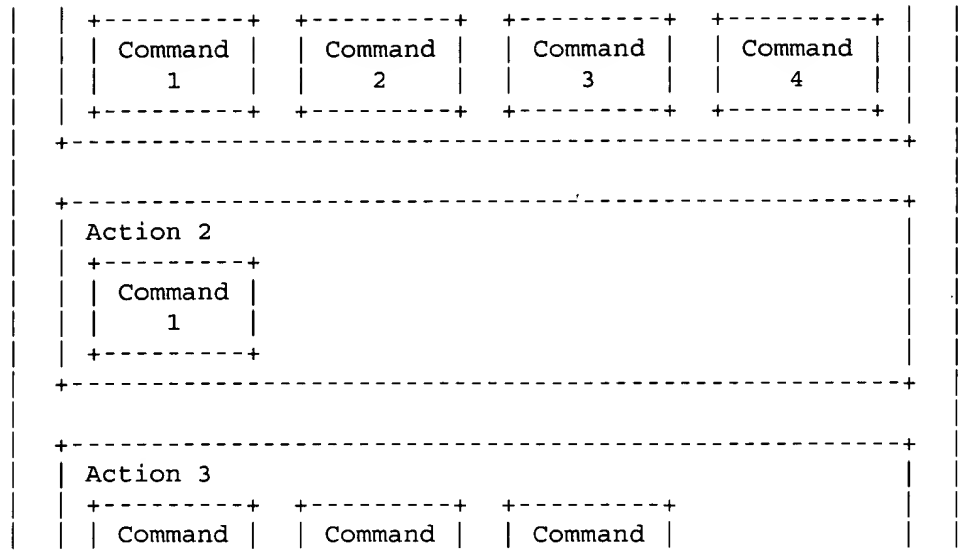
500 - Internal Gateway Error
501 - Not Implemented
502 - Not ready.
503 - Service Unavailable
504 - Command Received from unauthorized entity
505 - Command Received before Restart Response
510 - Insufficient resources
512 - Media Gateway unequipped to detect requested Event
513 - Media Gateway unequipped to generate requested Signals
514 - Media Gateway cannot send the specified announcement

515 - Unsupported Media Type
517 - Unsupported or invalid mode
518 - Event buffer full
519 - Out of space to store digit map
520 - Media Gateway does not have a digit map
521 - Termination is "ServiceChangeing"
526 - Insufficient bandwidth
529 - Internal hardware failure
530 - Temporary Network failure
531 - Permanent Network failure
581 - Does Not Exist

8. TRANSACTIONS

Commands between the Media Gateway Controller and the Media Gateway are grouped into Transactions, each of which is identified by a TransactionID. Transactions consist of one or more Actions. An Action consists of a series of Commands that are limited to operating within a single Context. Consequently each Action typically specifies a ContextID. However, there are two circumstances where a specific ContextID is not provided with an Action. One is the case of modification of a Termination outside of a Context. The other is where the controller requests the gateway to create a new Context. Following is a graphic representation of the Transaction, Action and Command relationships.





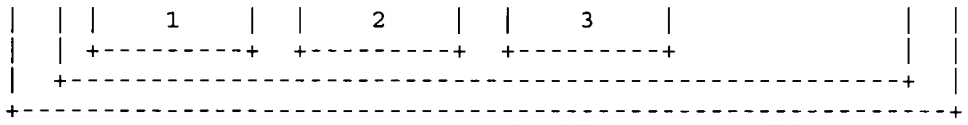


Figure 5 Transactions, Actions and Commands

Transactions are presented as TransactionRequests. Corresponding responses to a TransactionRequest are received in a single reply, possibly preceded by a number of TransactionPending messages (see section 8.2.3).

Transactions guarantee ordered Command processing. That is, Commands within a Transaction are executed sequentially. Ordering of Transactions is NOT guaranteed - transactions may be executed in any order, or simultaneously.

At the first failing Command in a Transaction, processing of the remaining Commands in that Transaction stops. If a command contains a wildcarded TerminationID, the command is attempted with each of the actual TerminationIDs matching the wildcard. A response within the TransactionReply is included for each matching TerminationID, even if one or more instances generated an error. If any TerminationID matching a wildcard results in an error when executed, any commands following the wildcarded command are not attempted. Commands may be marked as "Optional" which can override this behaviour - if a command marked as Optional results in an error, subsequent commands in the Transaction will be executed.

The TransactionReply includes the return values for the Commands that were executed successfully, and the Command and error descriptor for any Command that failed. TransactionPending is used to periodically notify the receiver that a Transaction has not completed yet, but is actively

being processed.

Applications SHOULD implement an application level timer per transaction. Expiration of the timer should cause a retransmission of the request. Receipt of a Reply should cancel the timer. Receipt of Pending should restart the timer.

8.1. Common Parameters

8.1.1. Transaction Identifiers

Transactions are identified by a TransactionID, which is assigned by sender and is unique within the scope of the sender.

8.1.2. Context Identifiers

Contexts are identified by a ContextID, which is assigned by the Media Gateway and is unique within the scope of the Media Gateway. The Media Gateway Controller shall use the ContextID supplied by the Media Gateway in all subsequent Transactions relating to that Context. The protocol makes reference to a distinguished value that may be used by the Media Gateway Controller when referring to a Termination that is currently not associated with a Context, namely the null ContextID.

The CHOOSE wildcard is used to request that the Media Gateway create a new Context. The MGC shall not use partially specified ContextIDs containing the CHOOSE wildcard. The MGC may use the ALL wildcard to address all Contexts on the MG.

8.2. Transaction Application Programming Interface

Following is an Application Programming Interface (API) describing the Transactions of the protocol. This API is shown to illustrate the Transactions and their parameters and is not intended to specify implementation (e.g. via use of blocking function calls). It will describe the input parameters and return values expected to be used by the various Transactions of the protocol from a very high level. Transaction syntax and encodings are specified in later subsections.

8.2.1. TransactionRequest

The TransactionRequest is invoked by the sender. There is one Transaction per request invocation. A request contains one or more Actions, each of which specifies its target Context and one or more Commands per Context.

```
TransactionRequest(TransactionId {
       ContextID {Command ... Command},
                      . . .
       ContextID  {Command ... Command } })
```

The TransactionID parameter must specify a value for later correlation with the TransactionReply or TransactionPending response from the receiver.

The ContextID parameter must specify a value to pertain to all Commands that follow up to either the next specification of a ContextID parameter or the end of the TransactionRequest, whichever comes first.

The Command parameter represents one of the Commands mentioned in the "Command Details" subsection titled "Application Programming Interface".

8.2.2. TransactionReply

The TransactionReply is invoked by the receiver. There is one reply invocation per transaction. A reply contains one or more Actions, each of which must specify its target Context and one or more Responses per Context.

```
TransactionReply(TransactionID {
      ContextID { Response ... Response },
      . . .
      ContextID { Response ... Response } })
```

The TransactionID parameter must be the same as that of the corresponding TransactionRequest.

The ContextID parameter must specify a value to pertain to all Responses for the action. The ContextID may be specific or null.

Each of the Response parameters represents a return value as mentioned in section 7.2, or an error descriptor if the command execution encountered an error. Commands after the point of failure are not processed and, therefore, Responses are not issued for them.

An exception to this occurs if a command has been marked as optional in the Transaction request. If the optional command generates an error, the transaction still continues to execute, so the Reply would, in this case, have Responses after an Error.

If the receiver encounters an error in processing a ContextID, the requested Action response will consist of the context ID and a single error descriptor, 422 Syntax Error in Action.

If the receiver encounters an error such that it cannot determine a legal Action, it will return a TransactionReply consisting of the TransactionID and a single error descriptor, 422 Syntax Error in Action. If the end of an action cannot be reliably determined but one or more Actions can be parsed, it will process them and then send 422 Syntax Error in Action as the last action for the transaction.If the receiver

encounters an error such that is cannot determine a legal Transaction, it will return a TransactionReply with a null TransactionID and a single error descriptor (403 Syntax Error in Transaction).

If the end of a transaction can not be reliably determined and one or more Actions can be parsed, it will process them and then return 403 Syntax Error in Transaction as the last action reply for the transaction. If no Actions can be parsed, it will return 403 Syntax Error in Transaction as the only reply

If the terminationID cannot be reliably determined it will send 442 Syntax Error in Command as the action reply.

If the end of a command cannot be reliably determined it will return 442 Syntax Error in Transaction as the reply to the last action it can parse.

8.2.3. TransactionPending

The receiver invokes the TransactionPending. A TransactionPending indicates that the Transaction is actively being processed, but has not been completed. It is used to prevent the sender from assuming the TransactionRequest was lost where the Transaction will take some time to complete.

TransactionPending(TransactionID { })

The TransactionID parameter must must be the same as that of the corresponding TransactionRequest. A property of root (normalMGExecutionTime) is settable by the MGC to indicate the interval within which the MGC expects a response to any transaction from the MG. Another property (normalMGCExecutionTime) is settable by the MGC to indicate the interval within which the MG should expects a response to any transaction from the MGC. Senders may receive more than one TransactionPending for a command. If a duplicate request is received when pending, the responder may send a duplicate pending immediately, or continue waiting for its timer to trigger another Transaction Pending.

8.3. Messages

Multiple Transactions can be concatenated into a Message. Messages have a header, which includes the identity of the sender. The Message Identifier (MID) of a message is set to a provisioned name (e.g. domain address/domain name/device name) of the entity transmitting the message. Domain name is a suggested default.

Every Message contains a Version Number identifying the version of the protocol the message conforms to. Versions consist of one or two digits, beginning with version 1 for the present version of the protocol.

The transactions in a message are treated independently. There is no order implied, there is no application or protocol acknowledgement of a message.

9. TRANSPORT

The transport mechanism for the protocol should allow the reliable transport of transactions between an MGC and MG. The transport shall remain independent of what particular commands are being sent and shall be applicable to all application states. There are several transports defined for the protocol, which are defined in normative Annexes to this document. Additional Transports may be defined as additional annexes in subsequent editions of this document, or in separate documents. For transport of the protocol over IP, MGCs shall implement both TCP and UDP/ALF, an MG shall implement TCP or UDP/ALF or both.

The MG is provisioned with a name or address (such as DNS name or IP address) of a primary and zero or more secondary MGCs (see section 7.2.8) that is the address the MG uses to send messages to the MGC. If TCP or UDP is used as the protocol transport and the port to which the initial ServiceChange request is to be sent is not otherwise known, that request should be sent to the default port number for the protocol. This port number is xxxx for text-encoded operation or yyyy for binary-encoded operation, for either UDP or TCP. The MGC receives the message containing the ServiceChange request from the MG and can determine the MG's address from it. As described in section 7.2.8, either the MG or the MGC may supply an address in the ServiceChangeAddress parameter to which subsequent transaction requests must be addressed, but responses (including the response to the initial ServiceChange request) must always be sent back to the address which was the source of the corresponding request.

9.1. Ordering of Commands

This document does not mandate that the underlying transport protocol guarantees the sequencing of transactions sent to an entity. This property tends to maximize the timeliness of actions, but it has a few drawbacks. For example:

* Notify commands may be delayed and arrive at the MGC after the transmission of a new command changing the EventsDescriptor

* If a new command is transmitted before a previous one is acknowledged, there is no guarantee that prior command will be executed before the new one.

Media Gateway Controllers that want to guarantee consistent operation of the Media Gateway may use the following rules:

1. When a Media Gateway handles several Terminations, commands pertaining to the different Terminations may be sent in parallel, for example following a model where each Termination (or group of

Terminations) is controlled by its own process or its own thread.

2. In a given Context, there should normally be at most one outstanding command (Add or Modify or Move). However, a Subtract command may be issued at any time. In consequence, a Media Gateway may sometimes receive a Modify command that applies to a previously subtracted Termination. Such commands should be ignored, and an error code should be returned.

3. On a given Termination, there should normally be at most one outstanding Notify command at any time. The RequestId parameter should be used to correlate Notify commands with the triggering notification request.

4. In some cases, an implicitly or explicitly wildcarded Subtract command that applies to a group of Terminations may step in front of a pending Add command. The Media Gateway Controller should individually delete all connections whose completion was pending at the time of the global Subtract command. Also, new Add commands for Terminations named by the wild-carding (or implied in a Multiplex descriptor) may not be sent until the wild-carded Subtract command is acknowledged.

5. AuditValue and AuditCapability are not subject to any sequencing.

6. ServiceChange shall always be the first command sent by a MG as defined by the restart procedure. Any other command or response must be delivered after this ServiceChange command. These rules do not affect the command responder, which should always respond to commands.

9.2. Protection against Restart Avalanche

In the event that a large number of Media Gateways are powered on simultaneously and they were to all initiate a ServiceChange transaction, the Media Gateway Controller would very likely be swamped, leading to message losses and network congestion during the critical period of service restoration. In order to prevent such avalanches, the following behavior is suggested:

1. When a Media Gateway is powered on, it should initiate a restart timer to a random value, uniformly distributed between 0 and a maximum waiting delay (MWD). Care should be taken to avoid synchronicity of the random number generation between multiple Media Gateways that would use the same algorithm.

2. The Media Gateway should then wait for either the end of this timer or the detection of a local user activity, such as for example an

off-hook transition on a residential Media Gateway.

3. When the timer elapses, or when an activity is detected, the Media Gateway should initiate the restart procedure.

The restart procedure simply requires the MG to guarantee that the first message that the Media Gateway Controller sees from this MG is a ServiceChange message informing the Media Gateway Controller about the restart

The value of MWD is a configuration parameter that depends on the type of the Media Gateway. The following reasoning may be used to determine the value of this delay on residential gateways.

Media Gateway Controllers are typically dimensioned to handle the peak hour traffic load, during which, in average, 10% of the lines will be busy, placing calls whose average duration is typically 3 minutes. The processing of a call typically involves 5 to 6 Media Gateway Controller transactions between each Media Gateway and the Media Gateway Controller. This simple calculation shows that the Media Gateway Controller is expected to handle 5 to 6 transactions for each Termination, every 30 minutes on average, or, to put it otherwise, about one transaction per Termination every 5 to 6 minutes on average. This suggests that a reasonable value of MWD for a residential gateway would be 10 to 12 minutes. In the absence of explicit configuration, residential gateways should adopt a value of 600 seconds for MWD.

The same reasoning suggests that the value of MWD should be much shorter for trunking gateways or for business gateways, because they handle a large number of Terminations, and also because the usage rate of these Terminations is much higher than 10% during the peak busy hour, a typical value being 60%. These Terminations, during the peak hour, are this expected to contribute about one transaction per minute to the Media Gateway Controller load. A reasonable algorithm is to make the value of MWD per "trunk" Termination six times shorter than the MWD per residential gateway, and also inversely proportional to the number of Terminations that are being restarted. For example MWD should be set to 2.5 seconds for a gateway that handles a T1 line, or to 60 milliseconds for a gateway that handles a T3 line.

10. SECURITY CONSIDERATIONS

This section covers security when using the protocol in an IP environment.

10.1. Protection of Protocol Connections

A security mechanism is clearly needed to prevent unauthorized entities

from using the protocol defined in this document for setting up unauthorized calls or interfering with authorized calls. The security mechanism for the protocol when transported over IP networks is IPsec [RFC2401 to RFC2411].

The AH header [RFC2402] affords data origin authentication, connectionless integrity and optional anti-replay protection of messages passed between the MG and the MGC. The ESP header [RFC2406] provides confidentiality of messages, if desired. For instance, the ESP encryption service should be requested if the session descriptions are used to carry session keys, as defined in SDP.

Implementations of the protocol defined in this document employing the ESP header SHALL comply with section 5 of [RFC2406], which defines a minimum set of algorithms for integrity checking and encryption. Similarly, implementations employing the AH header SHALL comply with section 5 of [RFC2402], which defines a minimum set of algorithms for integrity checking using manual keys.

Implementations SHOULD use IKE [RFC2409] to permit more robust keying options. Implementations employing IKE SHOULD support authentication with RSA signatures and RSA public key encryption.

10.2. Interim AH scheme

Implementation of IPsec requires that the AH or ESP header be inserted immediately after the IP header. This cannot be easily done at the application level. Therefore, this presents a deployment problem for the protocol defined in this document where the underlying network implementation does not support IPsec.

As an interim solution, an optional AH header is defined within the MEGACO protocol header. The header fields are exactly those of the SPI, SEQUENCE NUMBER and DATA fields as defined in [RFC2402]. The semantics of the header fields are the same as the "transport mode" of [RFC2402], except for the calculation of the Integrity Check value (ICV). In IPsec, the ICV is calculated over the entire IP packet including the IP header. This prevents spoofing of the IP addresses. To retain the same functionality, the ICV calculation should be performed across the entire transaction prepended by a synthesized IP header consisting of a 32 bit source IP address, a 32 bit destination address and an 16 bit UDP encoded as 10 hex digits. When the interim AH mechanism is employed when TCP is the transport Layer, the UDP Port above becomes the TCP port, and all other operations are the same.

Implementations of the MEGACO protocol SHALL implement IPsec where the underlying operating system and the transport network supports IPsec. Implementations of the protocol using IPv4 SHALL implement the interim

AH scheme. However, this interim scheme SHALL NOT be used when the underlying network layer supports IPsec. IPv6 implementations are assumed to support IPsec and SHALL NOT use the interim AH scheme.

All implementations of the interim AH mechanism SHALL comply with section 5 of [RFC2402] which defines a minimum set of algorithms for integrity checking using manual keys.

The interim AH interim scheme does not provide protection against eavesdropping; thus forbidding third parties from monitoring the connections set up by a given termination. Also, it does not provide protection against replay attacks. These procedures do not necessarily protect against denial of service attacks by misbehaving MGs or misbehaving MGCs. However, they will provide an identification of these misbehaving entities, which should then be deprived of their authorization through maintenance procedures.

10.3. Protection of Media Connections

The protocol allows the MGC to provide MGs with "session keys" that can be used to encrypt the audio messages, protecting against eavesdropping.

A specific problem of packet networks is "uncontrolled barge-in". This attack can be performed by directing media packets to the IP address and UDP port used by a connection. If no protection is implemented, the packets must be decompressed and the signals must be played on the "line side".

A basic protection against this attack is to only accept packets from known sources, checking for example that the IP source address and UDP source port match the values announced in the Remote Descriptor. This has two inconveniences: it slows down connection establishment and it can be fooled by source spoofing:

* To enable the address-based protection, the MGC must obtain the remote session description of the egress MG and pass it to the ingress MG. This requires at least one network roundtrip, and leaves us with a dilemma: either allow the call to proceed without waiting for the round trip to complete, and risk for example, "clipping" a remote announcement, or wait for the full roundtrip and settle for slower call-set-up procedures.

* Source spoofing is only effective if the attacker can obtain valid pairs of source destination addresses and ports, for example by listening to a fraction of the traffic. To fight source spoofing, one could try to control all access points to the network. But this is in practice very hard to achieve.

An alternative to checking the source address is to encrypt and authenticate the packets, using a secret key that is conveyed during the call set-up procedure. This will not slow down the call set- up, and provides strong protection against address spoofing.

11. MG-MGC CONTROL INTERFACE

The control association between MG and MGC is initiated at MG cold start, and announced by a ServiceChange message, but can be changed by subsequent events, such as failures or manual service events. While the protocol does not have an explicit mechanism to support multiple MGCs controlling a physical MG, it has been designed to support the multiple logical MG (within a single physical MG) that can be associated with different MGCs.

11.1. Multiple Virtual MGs

A physical Media Gateway may be partitioned into one or more Virtual MGs. A virtual MG consists of a set of statically partitioned physical Terminations and/or sets of ephemeral Terminations. A physical Termination is controlled by one MGC. The model does not require that other resources be statically allocated, just Terminations. The mechanism for allocating Terminations to virtual MGs is a management method outside the scope of the protocol. Each of the virtual MGs appears to the MGC as a complete MG client.

A physical MG may have only one network interface, which must be shared across virtual MGs. In such a case, the packet/cell side Termination is shared. It should be noted however, that in use, such interfaces require an ephemeral instance of the Termination to be created per flow, and thus sharing the Termination is straightforward. This mechanism does lead to a complication, namely that the MG must always know which of its controlling MGCs should be notified if an event occurs on the interface.

In normal operation, the Virtual MG will be instructed by the MGC to create network flows (if it is the originating side), or to expect flow requests (if it is the terminating side), and no confusion will arise. However, if an unexpected event occurs, the Virtual MG must know what to do with respect to the physical resources it is controlling.

If recovering from the event requires manipulation of a physical interface's state, only one MGC should do so. These issues are resolved by allowing any of the MGCs to create EventsDescriptors to be notified of such events, but only one MGC can have read/write access to the physical interface properties; all other MGCs have read-only access. The management mechanism is used to designate which MGC has read/write capability, and is designated the Master MGC.

Each virtual MG has its own Root Termination. In most cases the values for the properties of the Root Termination are independently settable by each MGC. Where there can only be one value, the parameter is read-only to all but the Master MGC.

ServiceChange may only be applied to a Termination or set of Terminations partitioned to the Virtual MG or created (in the case of ephemeral Terminations) by that Virtual MG.

11.2. Cold Start

A MG is pre-provisioned by a management mechanism outside the scope of this protocol with a Primary and (optionally) an ordered list of Secondary MGCs. Upon a cold start of the MG, it will issue a ServiceChange command with a "Restart" method, on the Root Termination to its primary MGC. If the MGC accepts the MG, it will send a Transaction Accept, with the MGCIdToTry set to itself. If the MG receives a ServiceChangeMGCId not equal to the MGC it contacted, it sends a ServiceChange to the MGC specified in the ServiceChangeMGCId. It continues this process until it gets a controlling MGC to accept its registration, or it fails to get a reply. Upon failure to obtain a reply, either from the Primary MGC, or a designated successor, the MG tries its pre-provisioned Secondary MGCs, in order. If the MG is unable to establish a control relationship with any MGC, it shall wait a random amount of time as described in section 9.2 and then start contacting its primary, and if necessary, its secondary MGCs again.

It is possible that the reply to a ServiceChange with Restart will be lost, and a command will be received by the MG prior to the receipt of the ServiceChange response. The MG shall issue error 505 - Command Received before Restart Response.

11.3. Negotiation of Protocol Version

The first ServiceChange command from an MG shall contain the version number of the protocol supported by the MG in the ServiceChangeVersion parameter. Upon receiving such a message, if the MGC supports only a lower version, then the MGC shall send a ServiceChangeReply with the lower version and thereafter all the messages between MG and MGC shall conform to the lower version of the protocol. If the MG is unable to comply and it has established a transport connection to the MGC, it should close that connection. In any event, it should reject all subsequent requests from the MGC with Error 406 Version Not Supported.

If the MGC supports a higher version than the MG but is able to support the lower version proposed by the MG, it shall send a ServiceChangeReply with the lower version and thereafter all the messages between MG and MGC shall conform to the lower version of the protocol. If the MGC is

unable to comply, it shall reject the association, with Error 406 Version Not Supported.

Protocol version negotiation may also occur at "handoff" and "failover" ServiceChanges.

11.4. Failure of an MG

If a MG fails, but is capable of sending a message to the MGC, it sends a ServiceChange with an appropriate method (graceful or forced) and specifies the Root TerminationID. When it returns to service, it sends a ServiceChange with a "Restart" method.

Allowing the MGC to send duplicate messages to both MGs accommodates pairs of MGs that are capable of redundant failover of one of the MGs. Only the Working MG shall accept or reject transactions. Upon failover, the Primary MG sends a ServiceChange command with a "Failover" method and a "MG Impending Failure" reason. The MGC then uses the primary MG as the active MG. When the error condition is repaired, the Working MG can send a "ServiceChange" with a "Restart" method.

11.5. Failure of an MGC

If the MG detects a failure of its controlling MGC, it attempts to contact the next MGC on its pre-provisioned list. It starts its attempts at the beginning (Primary MGC), unless that was the MGC that failed, in which case it starts at its first Secondary MGC. It sends a ServiceChange message with a "Failover" method and a "MGC Impending Failure" reason.

In partial failure, or manual maintenance reasons, an MGC may wish to direct its controlled MGs to use a different MGC. To do so, it sends a ServiceChange method to the MG with a "HandOff" method, and its designated replacement in ServiceChangeMGCId. The MG should send a ServiceChange message with a "Handoff" method and a "MGC directed change" reason to the designated MGC. If it fails to get a reply, or fails to see an Audit command subsequently, it should behave as if its MGC failed, and start contacting secondary MGCs. If the MG is unable to establish a control relationship with any MGC, it shall wait a random amount of time as described in section 9.2 and then start contacting its primary, and if necessary, its secondary MGCs again.

No recommendation is made on how the MGCs involved in the Handoff maintain state information; this is considered to be out of scope of this recommendation. The MGC and MG may take the following steps when Handoff occurs. When the MGC initiates a HandOff, the handover should be transparent to Operations on the Media Gateway. Transactions can be executed in any order, and could be in progress when the ServiceChange is

executed. Accordingly, commands in progress continue, transaction replies are sent to the new MGC (after a new control association is established), and the MG should expect outstanding transaction replies from the new MGC. No new messages shall be sent to the new MGC until the control association is established. Repeated transaction requests shall be directed to the new MGC. The MG shall maintain state on all

terminations and contexts.

It is possible that the MGC could be implemented in such a way that a failed MGC is replaced by a working MGC where the identity of the new MGC is the same as the failed one. In such a case, ServiceChangeMGCId would be specified with the previous value and the MG shall behave as if the value was changed, and send a ServiceChange message, as above.

Pairs of MGCs that are capable of redundant failover can notify the controlled MGs of the failover by the above mechanism.

12. PACKAGE DEFINITION

The primary mechanism for extension is by means of Packages. Packages define additional Properties, Events, Signals and Statistics that may occur on Terminations.

Packages defined by IETF will appear in separate RFCs.

Packages relevant to H.323 systems are listed in an Annex to Recommendation H.323.

Packages defined by ITU-T will be described in Annexes to H.248.

1) A public document or a standard forum document, which can be referenced as the document that describes the package following the guideline above, should be specified.

2) The document shall specify the version of the Package that it describes.

3) The document should be available on a public web server and should have a stable URL. The site should provide a mechanism to provide comments and appropriate responses should be returned.

12.1. Guidelines for defining packages

Packages define Properties, Events, Signals, and Statistics.

Names of all such defined constructs shall consist of the PackageID (which uniquely identifies the package) and the ID of the item (which uniquely identifies the item in that package). In the text encoding the

two shall be separated by a forward slash ("/") character. Example: togen/playtone is the text encoding to refer to the play tone signal in the tone generation package.

A Package will contain the following sections:

12.1.1. Package Overall description of the package, specifying:

Package Name: only descriptive,

PackageID: Is an identifier Description:

Version: A new version of a package can only add additional Properties, Events, Signals, Statistics and new possible values for an existing parameter described in the original package. No deletions or modifications shall be allowed. A version is an integer in the range from 1 to 99.

Extends (Optional): A package may extend an existing package. The version of the original package must be specified. When a package extends another package it shall only add additional Properties, Events, Signals, Statistics and new possible values for an existing parameter described in the original package. An extended package shall not redefine or overload a name defined in the original package. Hence, if package B version 1 extends package A version 1, version 2 of B will not be able to extend the A version 2 if A version 2 defines a name already in B version 1.

12.1.2. Properties

Properties defined by the package, specifying:

Property Name: only descriptive.
PropertyID: Is an identifier
Description:
Type: One of:
String: UTF-8 string
Integer: 4 byte signed integer
Double: 8 byte signed integer
Character: Unicode UTF-8 encoding of a single letter.
Could be more than one octet.
Enumeration: One of a list of possible unique values
(See 12.3)
Sub-list: A list of several values from a list
Boolean
Possible Values:
Defined in: Which descriptor the property is defined in.

LocalControl is for stream dependent properties.
TerminationState is for stream independent properties.
Characteristics: Read / Write or both, and (optionally), global:
Indicates whether a property is read-only, or read-write, and if it is global. If Global is omitted, the property is not global. If a property is declared as global, the value of the property is shared by all terminations realizing the package.

12.1.3. Events

Events defined by the package, specifying:
Event name: only descriptive.
EventID: Is an identifier
Description:
EventsDescriptor Parameters:
Parameters used by the MGC to configure the event, and found in the EventsDescriptor. See section 12.2
ObservedEventsDescriptor Parameters:
Parameters returned to the MGC in Notify requests and in replies to command requests from the MGC that audit ObservedEventsDescriptor, and found in the ObservedEventsDescriptor. See section 12.2

12.1.4. Signals

Signals defined by the package, specifying:
Signal Name: only descriptive.
SignalID: Is an identifier. SignalID is used in a SignalsDescriptor
Description
SignalType: One of:
OO (On/Off)
TO (TimeOut)
BR (Brief)

Note: SignalType may be defined such that it is dependent on the value of one or more parameters. Signals that would be played with SignalType BR should have a default duration. The package has to define the default duration and signalType.

Duration: in hundredths of seconds
Additional Parameters: See section 12.2

12.1.5. Statistics

Statistics defined by the package, specifying:
Statistic name: only descriptive.
StatisticID: Is an identifier
StatisticID is used in a StatisticsDescriptor
Description
Units: unit of measure, e.g. milliseconds, packets

12.1.6. Procedures

Additional guidance on the use of the package.

12.2. Guidelines to defining Properties, Statistics and Parameters to

Events and Signals.

Parameter Name: only descriptive
ParameterID: Is an identifier
Type: One of:
String: UTF-8 octet string
Integer: 4 octet signed integer
Double: 8 octet signed integer
Character: Unicode UTF-8 encoding of a single letter.
Could be more than one octet.
Enumeration: One of a list of possible unique
values (See 12.3)
Sub-list: A list of several values from a list
Boolean
Possible values:
Description:

12.3. Lists

Possible values for parameters include enumerations. Enumerations may be defined in a list. It is recommended that the list be IANA registered so that packages that extend the list can be defined without concern for conflicting names.

12.4. Identifiers

Identifiers in text encoding shall be strings of up to 64 characters, containing no spaces, starting with an alphanumeric character and consisting of alphanumeric characters and / or digits, and possibly including the special character underscore ("_"). Identifiers in binary encoding are 2 octets long. Both text and binary values shall be specified for each identifier, including identifiers used as values in

enumerated types.

12.5. Package Registration

A package can be registered with IANA for interoperability reasons. See section 13 for IANA considerations.

13. IANA CONSIDERATIONS

13.1. Packages

The following considerations SHALL be met to register a package with IANA:

1. A unique string name, unique serial number and version number is registered for each package. The string name is used with text encoding. The serial number shall be used with binary encoding. Serial Numbers 60000-64565 are reserved for private use. Serial

number 0 is reserved.

2. A contact name, email and postal addresses for that contact shall be specified. The contact information shall be updated by the defining organization as necessary.

3. A reference to a document that describes the package, which should be public: The document shall specify the version of the Package that it describes. If the document is public, it should be located on a public web server and should have a stable URL. The site should provide a mechanism to provide comments and appropriate responses should be returned.

4. Packages registered by other than recognized standards bodies shall have a minimum package name length of 8 characters

5. All other package names are first come-first served if all other conditions are met

13.2. Error Codes

The following considerations SHALL be met to register an error code with IANA:

1. An error number and a one line (80 character maximum) string is registered for each error.

2. A complete description of the conditions under which the error is detected shall be included in a publicly available document. The description shall be sufficiently clear to differentiate the error

from all other existing error codes.

3. The document should be available on a public web server and should have a stable URL.

4. Error numbers registered by recognized standards bodies shall have 3 or 4 character error numbers.

5. Error numbers registered by all other organizations or individuals shall have 4 character error numbers.

6. An error number shall not be redefined, nor modified except by the organization or individual that originally defined it, or their successors or assigns.

13.3. ServiceChange Reasons

The following considerations SHALL be met to register service change reason with IANA:

1. A one phrase, 80-character maximum, unique reason code is

registered for each reason.

2. A complete description of the conditions under which the reason is used is detected shall be included in a publicly available document. The description shall be sufficiently clear to differentiate the reason from all other existing reasons.

3. The document should be available on a public web server and should have a stable URL.

14. CONTACT INFORMATION

IETF Editor
Brian Rosen
Marconi
1000 FORE Drive
Warrendale, PA 15086
U.S.A.
Phone: +1 724-742-6826
Email: brosen@fore.com

ITU Editor
John Segers
Lucent Technologies
Room HE 306
Dept. Forward Looking Work
P.O. Box 18, 1270 AA Huizen

Netherlands
Phone: +31 35 687 4724
Email: jsegers@lucent.com

Additional IETF Authors
Fernando Cuervo
Nortel Networks
P.O. Box 3511 Stn C Ottawa, ON, K1Y 4H7
Canada
Email: cuervo@nortelnetworks.com

Bryan Hill
Gotham Networks
15 Discovery Way
Acton, MA 01720
USA
Phone: +1 978-263-6890
Email: bhill@gothamnetworks.com

Christian Huitema
Telcordia Technologies
MCC 1J236B
445 South Street
Morristown, NJ 07960

U.S.A.
Phone: +1 973-829-4266
EMail: huitema@research.telcordia.com

Nancy Greene
Nortel Networks
P.O. Box 3511 Stn C
Ottawa, ON, K1Y 4H7
Canada
Phone: +1 514-271-7221
Email: ngreene@nortelnetworks.com

Abdallah Rayhan
Nortel Networks
P.O. Box 3511 Stn C
Ottawa, ON, K1Y 4H7
Canada
Phone: +1 613-763-9611
Email: arayhan@nortelnetworks.com

ANNEX A BINARY ENCODING OF THE PROTOCOL (NORMATIVE)

This Annex specifies the syntax of messages using the notation defined in ASN.1 [ITU-T Recommendation X.680 (1997): Information Technology - Abstract Syntax Notation One (ASN.1) - Specification of basic notation.]. Messages shall be encoded for transmission by applying the basic encoding rules specified in [ITU-T Recommendation X.690(1994) Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER)].

A.1. Coding of wildcards

The use of wildcards ALL and CHOOSE is allowed in the protocol. This allows a MGC to partially specify Termination IDs and let the MG choose from the values that conform to the partial specification. Termination IDs may encode a hierarchy of names. This hierarchy is provisioned. For instance, a TerminationID may consist of a trunk group, a trunk within the group and a circuit. Wildcarding must be possible at all levels. The following paragraphs explain how this is achieved.

The ASN.1 description uses octet strings of up to 8 octets in length for Termination IDs. This means that Termination IDs consist of at most 64 bits. A fully specified Termination ID may be preceded by a sequence of wildcarding fields. A wildcarding field is octet in length. Bit 7 (the most significant bit) of this octet specifies what type of wildcarding is invoked: if the bit value equals 1, then the ALL wildcard is used; if the bit value if 0, then the CHOOSE wildcard is used. Bit 6 of the

wildcarding field specifies whether the wildcarding pertains to one level in the hierarchical naming scheme (bit value 0) or to the level of the hierarchy specified in the wildcarding field plus all lower levels (bit value 1). Bits 0 through 5 of the wildcarding field specify the bit position in the Termination ID at which the starts.

We illustrate this scheme with some examples. Assume that Termination IDs are three octets long and that each octet represents a level in a hierarchical naming scheme. A valid Termination ID is

```
00000001 00011110 01010101.
```

Addressing ALL names with prefix 00000001 00011110 is done as follows:

```
wildcarding field: 10000111
Termination ID: 00000001 00011110 xxxxxxxx.
```

The values of the bits labeled "x" is irrelevant and shall be ignored by the receiver.

Indicating to the receiver that is must choose a name with 00011110 as the second octet is done as follows:

```
wildcarding fields: 00010111 followed by 00000111
Termination ID: xxxxxxxx 00011110 xxxxxxxx.
```

The first wildcard field indicates a CHOOSE wildcard for the level in the naming hierarchy starting at bit 23, the highest level in our assumed naming scheme. The second wildcard field indicates a CHOOSE wildcard for the level in the naming hierarchy starting at bit 7, the lowest level in our assumed naming scheme.

Finally, a CHOOSE-wildcarded name with the highest level of the name equal to 00000001 is specified as follows:

```
wildcard field: 01001111
Termination ID: 0000001 xxxxxxxx xxxxxxxx .
```

Bit value 1 at bit position 6 of the first octet of the wildcard field indicates that the wildcarding pertains to the specified level in the naming hierarchy and all lower levels.

Context IDs may also be wildcarded. In the case of Context IDs, however, specifying partial names is not allowed. Context ID 0x0 SHALL be used to indicate the NULL Context, Context ID 0xFFFFFFFE SHALL be used to indicate a CHOOSE wildcard, and Context ID 0xFFFFFFFF SHALL be used to indicate an ALL wildcard.

TerminationID 0xFFFFFFFFFFFFFFFF SHALL be used to indicate the ROOT Termination.

A.2. ASN.1 syntax specification

This section contains the ASN.1 specification of the H.248 protocol syntax.

NOTE - In case a transport mechanism is used that employs application level framing, the definition of Transaction below changes. Refer to the annex defining the transport mechanism for the definition that applies in that case.

NOTE - The ASN.1 specification below contains a clause defining TerminationIDList as a sequence of TerminationIDs. The length of this sequence SHALL be one. The SEQUENCE OF construct is present only to allow future extensions.


```
MEDIA-GATEWAY-CONTROL DEFINITIONS AUTOMATIC TAGS::=
BEGIN


MegacoMessage ::= SEQUENCE
{
        authHeader      AuthenticationHeader OPTIONAL,
        mess                    Message
}

AuthenticationHeader ::= SEQUENCE
{
        secParmIndex    SecurityParmIndex,
        seqNum            SequenceNum,
        ad              AuthData
}

SecurityParmIndex ::= OCTET STRING(SIZE(4))

SequenceNum       ::= OCTET STRING(SIZE(4))

AuthData          ::= OCTET STRING (SIZE (16..32))

Message ::= SEQUENCE
{
        version         INTEGER(0..99),
        mId             MId,    -- Name/address of message originator
        messageBody             CHOICE
        {
                messageError    ErrorDescriptor,
                transactions    SEQUENCE OF Transaction
        },
        ...
```

```
}

MId ::= CHOICE
{
        ip4Address                              IP4Address,
        ip6Address                              IP6Address,
        domainName                      DomainName,
        deviceName                      PathName,
mtpAddress                              OCTET STRING(SIZE(5)),
 -- Addressing structure of mtpAddress:
 --        15                 0
 --        |  PC         | NI |
 --           14 bits     2 bits
      ...
}
```



```
DomainName ::= SEQUENCE
{
name            IA5String,
-- The name starts with an alphanumeric digit followed by a
-- sequence of alphanumeric digits, hyphens and dots.  No two
-- dots shall occur consecutively.
portNumber      INTEGER(0..65535) OPTIONAL
}

IP4Address ::= SEQUENCE
{
        address OCTET STRING (SIZE(4)),
        portNumber      INTEGER(0..65535) OPTIONAL
}

IP6Address ::= SEQUENCE
{
address OCTET STRING (SIZE(16)),
        portNumber      INTEGER(0..65535) OPTIONAL
}

PathName ::= IA5String(SIZE (1..64))
-- See section A.3

Transaction ::= CHOICE
{
        transactionRequest      TransactionRequest,
        transactionPending      TransactionPending,
        transactionReply        TransactionReply,
        ...
}

TransactionId ::= INTEGER(0..4294967295)  -- 32 bit unsigned integer

TransactionRequest ::= SEQUENCE
{
```

```
        transactionId           TransactionId,
        actions                 SEQUENCE OF ActionRequest,
        ...
}

TransactionPending ::= SEQUENCE
{
        transactionId           TransactionId,
        ...
}

TransactionReply ::= SEQUENCE
```



```
{
        transactionId           TransactionId,
        transactionResult       CHOICE
        {
             transactionError   ErrorDescriptor,
             actionReplies      SEQUENCE OF ActionReply
        },
        ...
}


ErrorDescriptor ::= SEQUENCE
{
        errorCode       ErrorCode,
        errorText       ErrorText OPTIONAL
}

ErrorCode ::= INTEGER(0..65535)
-- See section 13 for IANA considerations w.r.t. error codes

ErrorText ::= IA5String

ContextID ::= INTEGER(0..4294967295)

-- Context NULL Value: 0
-- Context CHOOSE Value: 429467294 (0xFFFFFFFE)
-- Context ALL Value: 4294967295 (0xFFFFFFFF)


ActionRequest ::= SEQUENCE
{
        contextId                       ContextID,
        contextRequest          ContextRequest OPTIONAL,
        contextAttrAuditReq     ContextAttrAuditRequest OPTIONAL,
        commandRequests         SEQUENCE OF CommandRequest
}

ActionReply ::= SEQUENCE
{
```

```
        contextId               ContextID,
        errorDescriptor ErrorDescriptor OPTIONAL,
        contextReply    ContextRequest OPTIONAL,
        commandReply    SEQUENCE OF CommandReply
}

ContextRequest ::= SEQUENCE
{
        priority        INTEGER(0..15) OPTIONAL,
```



```
        emergency       BOOLEAN OPTIONAL,
        topologyReq     SEQUENCE OF TopologyRequest OPTIONAL,
        ...
}

ContextAttrAuditRequest ::= SEQUENCE
{
topology        NULL OPTIONAL,
        emergency       NULL OPTIONAL,
        priority        NULL OPTIONAL,
        ...
}

CommandRequest ::= SEQUENCE
{
        command         Command,
        optional                NULL OPTIONAL,
        wildcardReturn  NULL OPTIONAL,
        ...
}

Command ::= CHOICE
{
        addReq                  AmmRequest,
        moveReq                 AmmRequest,
        modReq                  AmmRequest,
        -- Add, Move, Modify requests have the same parameters
        subtractReq             SubtractRequest,
        auditCapRequest         AuditRequest,
        auditValueRequest       AuditRequest,
        notifyReq               NotifyRequest,
        serviceChangeReq        ServiceChangeRequest,
        ...
}

CommandReply ::= CHOICE
{
        addReply                AmmsReply,
        moveReply               AmmsReply,
        modReply                AmmsReply,
        subtractReply           AmmsReply,
        -- Add, Move, Modify, Subtract replies have the same parameters
```

```
        auditCapReply           AuditReply,
        auditValueReply         AuditReply,
        notifyReply             NotifyReply,
        serviceChangeReply      ServiceChangeReply,
        ...
}
```



```
TopologyRequest ::= SEQUENCE
{
        terminationFrom         TerminationID,
        terminationTo           TerminationID,
        topologyDirection       ENUMERATED
        {
                bothway(0),
                isolate(1),
                oneway(2)
        }
}

AmmRequest ::= SEQUENCE
{
        terminationID           TerminationIDList,
        mediaDescriptor         MediaDescriptor OPTIONAL,
        modemDescriptor         ModemDescriptor OPTIONAL,
        muxDescriptor           MuxDescriptor OPTIONAL,
        eventsDescriptor        EventsDescriptor OPTIONAL,
        eventBufferDescriptor   EventBufferDescriptor OPTIONAL,
        signalsDescriptor       SignalsDescriptor OPTIONAL,
        digitMapDescriptor      DigitMapDescriptor OPTIONAL,
        auditDescriptor         AuditDescriptor OPTIONAL,
        ...
}

AmmsReply ::= SEQUENCE
{
        terminationID   TerminationIDList,
        terminationAudit                TerminationAudit OPTIONAL
}

SubtractRequest ::= SEQUENCE
{
        terminationID                   TerminationIDList,
        auditDescriptor                 AuditDescriptor OPTIONAL
}

AuditRequest ::= SEQUENCE
{
        terminationID           TerminationID,
        auditDescriptor         AuditDescriptor
}

AuditReply ::= SEQUENCE
```

```
{
        terminationID           TerminationID,
        auditResult                     AuditResult


Cuervo, Hill, Greene, Huitema, Rayhan, Rosen, Segers            [Page 79]



Internet draft              MEGACO Protocol             February 8, 2000


}

AuditResult ::= CHOICE
{
        contextAuditResult      TerminationIDList,
        terminationAuditResult  TerminationAudit
}



AuditDescriptor ::= SEQUENCE
{
auditToken      BIT STRING
{
                                muxToken(0), modemToken(1), mediaToken(2),
eventsToken(3), signalsToken(4),
digitMapToken(5), statsToken(6),
observedEventsToken(7),
packagesToken(8), eventBufferToken(9)
} OPTIONAL,
        ...
}

TerminationAudit ::= SEQUENCE OF AuditReturnParameter

AuditReturnParameter ::= CHOICE
{
        errorDescriptor                 ErrorDescriptor,
mediaDescriptor                         MediaDescriptor,
        modemDescriptor                         ModemDescriptor,
        muxDescriptor                   MuxDescriptor,
        eventsDescriptor                EventsDescriptor,
        eventBufferDescriptor           EventBufferDescriptor,
        signalsDescriptor               SignalsDescriptor,
        digitMapDescriptor              DigitMapDescriptor,
        observedEventsDescriptor        ObservedEventsDescriptor,
        statisticsDescriptor            StatisticsDescriptor,
        packagesDescriptor              PackagesDescriptor,
        ...
}

NotifyRequest ::= SEQUENCE
{
        terminationID                   TerminationIDList,
        observedEventsDescriptor        ObservedEventsDescriptor,
        errorDescriptor                 ErrorDescriptor OPTIONAL
}
```

```
NotifyReply ::= SEQUENCE
{
        terminationID                           TerminationIDList OPTIONAL,
        errorDescriptor                         ErrorDescriptor OPTIONAL
}

ObservedEventsDescriptor ::= SEQUENCE
{
        requestId                               RequestID,
        observedEventLst                        SEQUENCE OF ObservedEvent
}

ObservedEvent ::= SEQUENCE
{
        eventName                               EventName,
        streamID                                StreamID OPTIONAL,
        eventParList                    SEQUENCE OF EventParameter,
        timeNotation                    TimeNotation OPTIONAL
}

EventName ::= PkgdName

EventParameter ::= SEQUENCE
{
        eventParameterName      Name,
        value                           Value
}

ServiceChangeRequest ::= SEQUENCE
{
        terminationID                   TerminationIDList,
        serviceChangeParms              ServiceChangeParm
}

ServiceChangeReply ::= SEQUENCE
{
        terminationID                   TerminationIDList,
        serviceChangeResult             ServiceChangeResult
}

-- For ServiceChangeResult, no parameters are mandatory.  Hence the
-- distinction between ServiceChangeParm and ServiceChangeResParm.

ServiceChangeResult ::= CHOICE
{
        errorDescriptor                 ErrorDescriptor,
        serviceChangeResParms           ServiceChangeResParm
}
```

```
WildcardField ::= OCTET STRING(SIZE(1))

TerminationID ::= SEQUENCE
{
        wildcard        SEQUENCE OF WildcardField,
        id              OCTET STRING(SIZE(1..8))
}
-- See Section A.1 for explanation of wildcarding mechanism.
-- Termination ID 0xFFFFFFFFFFFFFFFF indicates the ROOT Termination.

TerminationIDList ::= SEQUENCE OF TerminationID

MediaDescriptor ::= SEQUENCE
{

        termStateDescr  TerminationStateDescriptor OPTIONAL,
        streams         CHOICE
                                {
                                        oneStream       StreamParms,
                                        multiStream     SEQUENCE OF StreamDescriptor
                                },
        ...
}

StreamDescriptor ::= SEQUENCE
{
        streamID                        StreamID,
        streamParms                             StreamParms
}

StreamParms ::= SEQUENCE
{
        localControlDescriptor          LocalControlDescriptor OPTIONAL,
        localDescriptor                 LocalRemoteDescriptor OPTIONAL,
        remoteDescriptor                LocalRemoteDescriptor OPTIONAL,
        ...
}

LocalControlDescriptor ::= SEQUENCE
{
        streamMode      StreamMode OPTIONAL,
        reserve         BOOLEAN,
        propertyParms   SEQUENCE OF PropertyParm,
        ...
}

StreamMode ::= ENUMERATED
{
```

```
        sendOnly(0),
        recvOnly(1),
        sendRecv(2),
        inactive(3),
        loopBack(4),
        ...
}

-- In PropertyParm, value is a SEQUENCE OF octet string.  When sent
-- by an MGC the interpretation is as follows:
-- empty sequence means CHOOSE
-- one element sequence specifies value
-- longer sequence means "choose one of the values"
-- The relation field may only be selected if the value sequence
-- has length 1.  It indicates that the MG has to choose a value
-- for the property. E.g., x > 3 (using the greaterThan
-- value for relation) instructs the MG to choose any value larger
-- than 3 for property x.
-- The range field may only be selected if the value sequence
-- has length 2.  It indicates that the MG has to choose a value
-- in the range between the first octet in the value sequence and
-- the trailing octet in the value sequence, including the
-- boundary values.
-- When sent by the MG, only responses to an AuditCapability request
-- may contain multiple values, a range, or a relation field.

PropertyParm ::= SEQUENCE
{
        name            PkgdName,
        value   SEQUENCE OF OCTET STRING,
        extraInfo       CHOICE
        {
                relation        Relation,
                range           BOOLEAN
        } OPTIONAL

}

Name ::= OCTET STRING(SIZE(2))

PkgdName ::= OCTET STRING(SIZE(4))
-- represents Package Name (2 octets) plus Property Name (2 octets)
-- To wildcard a package use 0xFFFF for first two octets, choose
-- is not allowed. Wildcard of a package name is permitted only if
-- Property Name is also wildcarded. To reference native property
-- tag specified in Annex C, use 0x0000 as first two octets.

Relation ::= ENUMERATED
```

```
{
        greaterThan(0),
        smallerThan(1),
        unequalTo(2),
        ...
}

LocalRemoteDescriptor ::= SEQUENCE
{
        propGrps        SEQUENCE OF PropertyGroup,
        ...
}

PropertyGroup ::= SEQUENCE OF PropertyParm

TerminationStateDescriptor ::= SEQUENCE
{
        propertyParms       SEQUENCE OF PropertyParm,
        eventBufferFlag   BOOLEAN,
        serviceState        ServiceState OPTIONAL,
        ...
}

ServiceState ::= ENUMERATED
{
        test(0),
        outOfSvc(1),
        inSvc(2),
      ...
}

MuxDescriptor   ::= SEQUENCE
{
        muxType                 MuxType,
        termList        SEQUENCE OF TerminationID,
        ...
}

MuxType ::= ENUMERATED
{
        h221(0),
        h223(1),
        h226(2),
        v76(3),
        ...
}

StreamID ::= INTEGER(0..65535)  -- 16 bit unsigned integer
```

```
EventsDescriptor ::= SEQUENCE
{
        requestID               RequestID,
        eventList               SEQUENCE OF RequestedEvent
}

RequestedEvent ::= SEQUENCE
{
        pkgdName        PkgdName,
        streamID                StreamID OPTIONAL,
        eventAction     RequestedActions OPTIONAL,
        evParList       SEQUENCE OF EventParameter
}

RequestedActions ::= SEQUENCE
{
        keepActive                      BOOLEAN,
        eventDM                 EventDM OPTIONAL,
        secondEvent             SecondEventsDescriptor OPTIONAL,
        signalsDescriptor       SignalsDescriptor OPTIONAL,
        ...
}


EventDM ::= CHOICE
{       digitMapName    DigitMapName,
        digitMapValue   DigitMapValue
}

SecondEventsDescriptor ::= SEQUENCE
{
        requestID                       RequestID,
        eventList                       SEQUENCE OF SecondRequestedEvent
}

SecondRequestedEvent ::= SEQUENCE
{
        pkgdName                PkgdName,
        streamID                        StreamID OPTIONAL,
        eventAction             SecondRequestedActions OPTIONAL,
        evParList               SEQUENCE OF EventParameter
}

SecondRequestedActions ::= SEQUENCE
{
        keepActive                      BOOLEAN,
        eventDM                 EventDM OPTIONAL,
        signalsDescriptor       SignalsDescriptor OPTIONAL,
```



```
        ...
}
```

```
EventBufferDescriptor ::= SEQUENCE OF ObservedEvent

SignalsDescriptor ::= SEQUENCE OF SignalRequest

SignalRequest ::=CHOICE
{
        signal          Signal,
        seqSigList              SeqSigList
}

SeqSigList ::= SEQUENCE
{
        id              INTEGER(0..65535),
        signalList      SEQUENCE OF Signal
}

Signal ::= SEQUENCE
{
        signalName              SignalName,
        streamID                StreamID OPTIONAL,
        sigType         SignalType OPTIONAL,
        duration                INTEGER (0..65535) OPTIONAL,
        notifyCompletion        BOOLEAN,
        sigParList              SEQUENCE OF SigParameter
}

SignalType ::= ENUMERATED
{
        brief(0),
        onOff(1),
        timeOut(2),
        ...
}

SignalName ::= PkgdName

SigParameter ::= SEQUENCE
{
        sigParameterName                Name,
        value                           Value
}

RequestID ::= INTEGER(0..4294967295)    -- 32 bit unsigned integer

ModemDescriptor ::= SEQUENCE
```



```
{
mtl     SEQUENCE OF ModemType,
        mpl     SEQUENCE OF PropertyParm

}
```

```
ModemType ::= ENUMERATED
{
        v18(0),
        v22(1),
        v22bis(2),
        v32(3),
        v32bis(4),
        v34(5),
        v90(6),
        v91(7),
        synchISDN(8),
        ...
}

DigitMapDescriptor ::= SEQUENCE
{
        digitMapName    DigitMapName,
        digitMapValue   DigitMapValue
}

DigitMapName ::= Name

DigitMapValue ::= SEQUENCE
{
        startTimer              INTEGER(0..99) OPTIONAL,
        shortTimer              INTEGER(0..99) OPTIONAL,
        longTimer               INTEGER(0..99) OPTIONAL,
        digitMapBody    IA5String
        -- See Section A.3 for explanation of digit map syntax
}

ServiceChangeParm ::= SEQUENCE
{
        serviceChangeMethod     ServiceChangeMethod,
        serviceChangeAddress    ServiceChangeAddress OPTIONAL,
        serviceChangeVersion    INTEGER(0..99) OPTIONAL,
        serviceChangeProfile    ServiceChangeProfile OPTIONAL,
        serviceChangeReason     Value,
        serviceChangeDelay      INTEGER(0..4294967295) OPTIONAL,
                                    -- 32 bit unsigned integer
        serviceChangeMgcId      MId OPTIONAL,
        timeStamp                       TimeNotation OPTIONAL,
```



```
}

ServiceChangeAddress ::= CHOICE
{
        portNumber                      INTEGER(0..65535), -- TCP/UDP port number
        ip4Address                      IP4Address,
        ip6Address                      IP6Address,
        domainName              DomainName,
        deviceName              PathName,
```

```
mtpAddress                      OCTET STRING(SIZE(5)),
        ...
}

ServiceChangeResParm ::= SEQUENCE
{
        serviceChangeMgcId      MId OPTIONAL,
        serviceChangeAddress    ServiceChangeAddress OPTIONAL,
        serviceChangeVersion    INTEGER(0..99) OPTIONAL,
serviceChangeProfile    ServiceChangeProfile OPTIONAL
}

ServiceChangeMethod ::= ENUMERATED
{
        failover(0),
        forced(1),
        graceful(2),
        restart(3),
        disconnected(4),
        handOff(5),
        ...
}

ServiceChangeProfile ::= SEQUENCE
{
        profileName     Name,
        version         INTEGER(0..99)
}

PackagesDescriptor ::= SEQUENCE OF PackagesItem

PackagesItem ::= SEQUENCE
{
packageName             Name,
packageVersion  INTEGER(0..99)
}

StatisticsDescriptor ::= SEQUENCE OF StatisticsParameter
```



```
StatisticsParameter ::= SEQUENCE
{
        statName                PkgdName,
        statValue               Value
}

TimeNotation ::= SEQUENCE
{
        date            IA5String(SIZE(8)),  -- yyyymmdd format
        time            IA5String(SIZE(8))   -- hhmmssss format
}
```

```
Value ::= OCTET STRING


END
```

A.3. Digit maps and path names

>From a syntactic viewpoint, digit maps are strings with syntactic restrictions imposed upon them. The syntax of valid digit maps is specified in ABNF [RFC 2119]. The syntax for digit maps presented in this section is for illustrative purposes only. The definition of digitMap in Annex B takes precedence in the case of differences between the two.

```
digitMap = (digitString / LWSP "(" LWSP digitStringList LWSP
            ")" LWSP)
digitStringList = digitString *( LWSP "/" LWSP digitString )
digitString = 1*(digitStringElement)
digitStringElement = digitPosition [DOT]
digitPosition = digitMapLetter / digitMapRange
digitMapRange = ("x" / LWSP "[" LWSP digitLetter LWSP "]" LWSP)
digitLetter = *((DIGIT "-" DIGIT) / digitMapLetter)
digitMapLetter = DIGIT  /  %x41-4B / %x61-6B /     ; Mapped events 0-9,
                                                   ; A-K (a-k)
                 "L"  ; Long duration modifier
LWSP = *(WSP / COMMENT / EOL)
WSP = SP / HTAB
COMMENT = ";" *(SafeChar / RestChar / WSP) EOL
EOL = (CR [LF]) / LF
SP = %x20
HTAB = %x09
CR = &x0D
LF = %x0A
SafeChar = DIGIT / ALPHA / "+" / "-" / "&" / "!" / "_" / "/" /
"'" / "?" / "@" / "^" / "`" / "~" / "*" / "$" / "
```



```
"(" / ")" / "%" / "."
RestChar = ";" / "[" / "]" / "{" / "}" / ":" / "," / "#" /
               "<" / ">" / "=" / %x22
DIGIT = %x30-39                 ; digits 0 through 9
ALPHA = %x41-5A / %x61-7A       ; A-Z, a-z
```

A path name is also a string with syntactic restrictions imposed upon it. The ABNF production defining it is copied from Annex B.

```
PathName = NAME *(["/"] ["*"] ["@"] (ALPHA / DIGIT)) ["*"]
NAME = ALPHA *63(ALPHA / DIGIT / "_" )
```

ANNEX B TEXT ENCODING OF THE PROTOCOL (NORMATIVE)

B.1. Coding of wildcards

In a text encoding of the protocol, while TerminationIDs are arbitrary, by judicious choice of names, the wildcard character, "*" may be made more useful. When the wildcard character is encountered, it will "match" all TerminationIDs having the same previous and following characters (if appropriate). For example, if there were TerminationIDs of R13/3/1, R13/3/2 and R13/3/3, the TerminationID R13/3/* would match all of them. There are some circumstances where ALL Terminations must be referred to. The TerminationID "*" suffices, and is referred to as ALL. The CHOOSE TerminationID "$" may be used to signal to the MG that it has to create an ephemeral Termination or select an idle physical Termination.

B.2. ABNF specification

The protocol syntax is presented in ABNF according to RFC2234. The protocol is not case sensitive. Identifiers are not case sensitive.

```
megacoMessage        = LWSP [authenticationHeader SEP ] message

authenticationHeader = AuthToken EQUAL SecurityParmIndex COLON
                       SequenceNum COLON AuthData

SecurityParmIndex    = "0x" 8(HEXDIG)
SequenceNum          = "0x" 8(HEXDIG)
AuthData             = "0x" 32*64(HEXDIG)

message            = MegacopToken SLASH Version SEP mId SEP
```



```
messageBody

messageBody          = ( errorDescriptor / transactionList )

transactionList      = 1*( transactionRequest / transactionReply /
                       transactionPending)

transactionPending   = PendingToken EQUAL TransactionID LBRKT RBRKT

transactionRequest   = TransToken EQUAL TransactionID LBRKT
                       actionRequest *(COMMA actionRequest) RBRKT

actionRequest        = CtxToken EQUAL ContextID LBRKT ((
                       contextRequest [COMMA  commandRequestList])
                       / commandRequestList) RBRKT

contextRequest     = ((contextAudit [COMMA contextProperties])
                          / contextProperties)
```

```
contextProperties     = contextProperty *(COMMA contextProperty)

; at-most-once
contextProperty       = (topologyDescriptor / priority /
EmergencyToken)

contextAudit        = ContextAuditToken LBRKT
                        contextAuditProperties *(COMMA
                        contextAuditProperties) RBRKT

; at-most-once
contextAuditProperties = ( TopologyToken / EmergencyToken /
                           PriorityToken )

commandRequestList=["O-"] commandRequest *(COMMA ["O-"]commandRequest)

commandRequest        = ( ammRequest / subtractRequest / auditRequest /
                          notifyRequest / serviceChangeRequest)

transactionReply      = ReplyToken EQUAL TransactionID LBRKT
                        ( errorDescriptor / actionReplyList ) RBRKT

actionReplyList       = actionReply *(COMMA actionReply )

actionReply           = CtxToken EQUAL ContextID LBRKT
                        ( errorDescriptor / commandReply ) RBRKT

commandReply        = (( contextProperties [COMMA commandReplyList] ) /
                          commandReplyList )
```



```
commandReplyList      = commandReplys *(COMMA commandReplys )

commandReplys         = (serviceChangeReply / auditReply / ammsReply /
                          notifyReply )

;Add Move and Modify have the same request parameters
ammRequest            = (AddToken / MoveToken / ModifyToken ) EQUAL
                        TerminationID [LBRKT ammParameter *(COMMA
                        ammParameter) RBRKT]

;at-most-once
ammParameter          = (mediaDescriptor / modemDescriptor /
                          muxDescriptor / eventsDescriptor /
                          signalsDescriptor / digitMapDescriptor /
                          eventBufferDescriptor / auditDescriptor)

ammsReply             = (AddToken / MoveToken / ModifyToken /
                          SubtractToken ) EQUAL TerminationID [ LBRKT
                          terminationAudit RBRKT ]

subtractRequest       =  ["W-"] SubtractToken EQUAL TerminationID
```

```
                        [ LBRKT auditDescriptor RBRKT]

auditRequest          = ["W-"] (AuditValueToken / AuditCapToken ) EQUAL
                         TerminationID LBRKT auditDescriptor RBRKT

auditReply            = (AuditValueToken / AuditCapToken )
                        ( contextTerminationAudit  / auditOther)

auditOther            = EQUAL TerminationID LBRKT
                        terminationAudit RBRKT

terminationAudit  = auditReturnParameter *(COMMA auditReturnParameter)

contextTerminationAudit = EQUAL CtxToken ( terminationIDList /
                       LBRKT errorDescriptor RBRKT )
;at-most-once except errorDescriptor
auditReturnParameter = (mediaDescriptor / modemDescriptor /
                         muxDescriptor / eventsDescriptor /
                         signalsDescriptor / digitMapDescriptor /
                     observedEventsDescriptor / eventBufferDescriptor /
                         statisticsDescriptor / packagesDescriptor /
                          errorDescriptor )

auditDescriptor       = AuditToken LBRKT [ auditItem
                        *(COMMA auditItem) ] RBRKT

notifyRequest         = NotifyToken EQUAL TerminationID
```



```
                       LBRKT ( observedEventsDescriptor /
                             errorDescriptor ) RBRKT

notifyReply           = NotifyToken EQUAL TerminationID
                        [ LBRKT errorDescriptor RBRKT ]

serviceChangeRequest = ServiceChangeToken EQUAL TerminationID
                       LBRKT serviceChangeDescriptor RBRKT

serviceChangeReply    = ServiceChangeToken EQUAL TerminationID
                        [LBRKT (errorDescriptor /
                        serviceChangeReplyDescriptor) RBRKT]

errorDescriptor    = ErrorToken EQUAL ErrorCode
                     LBRKT [quotedString] RBRKT

ErrorCode             = 1*4(DIGIT) ; could be extended

TransactionID         = UINT32

mId                   = (( domainAddress / domainName )
                        [":" portNumber]) / mtpAddress / deviceName

; ABNF allows two or more consecutive "." although it is meaningless
```

```
; in a domain name.
domainName            = "<" (ALPHA / DIGIT) *63(ALPHA / DIGIT / "-" /
                        ".") ">"
deviceName            = pathNAME

ContextID             = (UINT32 / "*" / "-" / "$")

domainAddress         = "[" (IPv4address / IPv6address) "]"
;RFC2373 contains the definition of IP6Addresses.
IPv6address           = hexpart [ ":" IPv4address ]
IPv4address           = V4hex DOT V4hex DOT V4hex DOT V4hex
V4hex                 = 1*3(DIGIT) ; "0".."225"
; this production, while occurring in RFC2373, is not referenced
; IPv6prefix            = hexpart SLASH 1*2DIGIT
hexpart              = hexseq "::" [ hexseq ] / "::" [ hexseq ] /
hexseq
hexseq                = hex4 *( ":" hex4)
hex4                  = 1*4HEXDIG

portNumber            = UINT16

; An mtp address is five octets long
mtpAddress            = MTPToken LBRKT octetString RBRKT
```



```
terminationIDList     = LBRKT TerminationID *(COMMA TerminationID)
RBRKT

; Total length of pathNAME must not exceed 64 chars.
pathNAME          = ["*"] NAME *("/" / "*"/ ALPHA / DIGIT /"_" / "$" )
                    ["@" pathDomainName ]

; ABNF allows two or more consecutive "." although it is meaningless
; in a path domain name.
pathDomainName        = (ALPHA / DIGIT / "*" )
                        *63(ALPHA / DIGIT / "-" / "*" / ".")

TerminationID         = "ROOT" / pathNAME / "$" / "*"

mediaDescriptor   = MediaToken LBRKT mediaParm *(COMMA mediaParm) RBRKT

; at-most-once per item
; and either streamParm or streamDescriptor but not both
mediaParm             = (streamParm / streamDescriptor /
                         terminationStateDescriptor)

; at-most-once
streamParm            = ( localDescriptor / remoteDescriptor /
                         localControlDescriptor )

streamDescriptor      = StreamToken EQUAL StreamID LBRKT streamParm
                        *(COMMA streamParm) RBRKT
```

```
localControlDescriptor = LocalControlToken LBRKT localParm
                         *(COMMA localParm) RBRKT

; at-most-once per item
localParm             = ( streamMode / propertyParm / reservedMode)

reservedMode        = ReservedToken EQUAL ( "ON" / "OFF" )

streamMode            = ModeToken EQUAL streamModes

streamModes          = (SendonlyToken / RecvonlyToken / SendrecvToken /
                        InactiveToken / LoopbackToken )

propertyParm          = pkgdName parmValue
parmValue             = (EQUAL alternativeValue/ INEQUAL VALUE)
alternativeValue      = ( VALUE / LSBRKT VALUE *(COMMA VALUE) RSBRKT  /
                        LSBRKT VALUE DOT DOT VALUE RSBRKT )

INEQUAL               = LWSP (">" / "<" / "#" ) LWSP
LSBRKT                = LWSP "[" LWSP
```



```
RSBRKT                = LWSP "]" LWSP

localDescriptor       = LocalToken LBRKT octetString RBRKT

remoteDescriptor      = RemoteToken LBRKT octetString RBRKT

eventBufferDescriptor= EventBufferToken LBRKT observedEvent
                        *( COMMA observedEvent ) RBRKT

eventBufferFlag       = BufferToken EQUAL ("ON" / "OFF" )

terminationStateDescriptor = TerminationStateToken LBRKT
            terminationStateParm *( COMMA terminationStateParm ) RBRKT

; at-most-once per item
terminationStateParm=(propertyParm / serviceStates / eventBufferFlag )

serviceStates         = ServiceStatesToken EQUAL ( TestToken /
                        OutOfSvcToken / InSvcToken )

muxDescriptor         = MuxToken EQUAL MuxType  terminationIDList

MuxType               = ( H221Token / H223Token / H226Token / V75Token
                         extensionParameter )

StreamID              = UINT16
pkgdName              = (PackageName SLASH ItemID) /    ; specific item
                        (PackageName SLASH "*" ) /  ; all events in pkg
                        ( "*" SLASH "*" )  ; all events supported by MG
PackageName           = NAME
```

```
ItemID                = NAME

eventsDescriptor      = EventsToken EQUAL RequestID LBRKT
                        requestedEvent *( COMMA requestedEvent ) RBRKT

requestedEvent        = pkgdName [ LBRKT eventParameter
                        *( COMMA eventParameter ) RBRKT ]

; at-most-once each of embedOrKeepActive , eventDM or eventStream
eventParameter        = ( embedOrKeepActive / eventDM / eventStream
                         / eventOther )
embedOrKeepActive     = eventEmbedded / KeepActiveToken

eventEmbedded = EmbedToken LBRKT embedFirst [COMMA embedFirst ] RBRKT
; at-most-once of each
embedFirst      = ( signalsDescriptor / secondEventEmbeddedDescriptor )

secondEventEmbeddedDescriptor= EventsToken EQUAL RequestID LBRKT
```



```
              secondRequestedEvent *(COMMA secondRequestedEvent) RBRKT

secondRequestedEvent = pkgdName [ LBRKT secondEventParameter
                       *( COMMA secondEventParameter ) RBRKT ]

; at-most-once each of embedOrKeepActive , eventDM or eventStream
secondEventParameter = ( SecondEmbedOrKeepActive / eventDM /
                         eventStream / eventOther )

secondEmbedOrKeepActive = secondEventEmbedded / KeepActiveToken

secondEventEmbedded  = EmbedToken LBRKT signalsDescriptor RBRKT

eventStream          = StreamToken EQUAL StreamID

eventOther           = eventParameterName parmValue

eventParameterName   = NAME

eventDM              = DigitMapToken ((EQUAL digitMapName ) /
                       (LBRKT digitMapValue RBRKT ))

signalsDescriptor    = SignalsToken LBRKT [ signalParm
                       *(COMMA signalParm)] RBRKT

signalParm           = signalList / signalRequest

signalRequest        = signalName [ LBRKT sigParameter
                       *(COMMA sigParameter) RBRKT ]

signalList           = SignalListToken EQUAL signalListId LBRKT
                       signalListParm *(COMMA signalListParm) RBRKT
```

```
signalListId          = UINT16

;exactly once signalType, at most once duration and every signal
;parameter
signalListParm        = signalRequest

signalName            = pkgdName
;at-most-once sigStream, at-most-once sigSignalType,
;at-most-once sigDuration, every signalParameterName at most once
sigParameter        = sigStream / sigSignalType / sigDuration /
sigOther
                          / notifyCompletion
sigStream             = StreamToken EQUAL StreamID
sigOther              = sigParameterName parmValue
sigParameterName      = NAME
```



```
sigSignalType         = SignalTypeToken EQUAL signalType
signalType            = (OnOffToken / TimeOutToken / BriefToken)
sigDuration           = DurationToken EQUAL UINT16
notifyCompletion      = NotifyCompletionToken EQUAL ("ON" / "OFF" )

observedEventsDescriptor = ObservedEventsToken EQUAL RequestID
                    LBRKT observedEvent *(COMMA observedEvent) RBRKT

;time per event, because it might be buffered
observedEvent         = [ TimeStamp LWSP COLON] LWSP
                        pkgdName [ LBRKT observedEventParameter
                        *(COMMA observedEventParameter) RBRKT ]

;at-most-once eventStream, every eventParameterName at most once
observedEventParameter = eventStream / eventOther

RequestID             = UINT32

modemDescriptor       = ModemToken (( EQUAL modemType) /
                        (LSBRKT modemType *(COMMA modemType) RSBRKT))
                        [ LBRKT NAME parmValue
                       *(COMMA NAME parmValue) RBRKT ]

; at-most-once
modemType             = (V32bisToken / V22bisToken / V18Token /
                         V22Token / V32Token / V34Token / V90Token /
                        V91Token / SynchISDNToken / extensionParameter)

digitMapDescriptor    = DigitMapToken EQUAL digitMapName
                        ( LBRKT digitMapValue RBRKT )
digitMapName          = NAME
digitMapValue         = ["T" COLON Timer COMMA] ["S" COLON Timer COMMA]
                        ["L" COLON Timer COMMA] digitMap
Timer                 = 1*2DIGIT
digitMap = (digitString / LWSP "(" LWSP digitStringList LWSP ")" LWSP)
digitStringList       = digitString *( LWSP "|" LWSP digitString )
```

```
digitString          = 1*(digitStringElement)
digitStringElement   = digitPosition [DOT]
digitPosition        = digitMapLetter / digitMapRange
digitMapRange        = ("x" / LWSP "[" LWSP digitLetter LWSP "]" LWSP)
digitLetter          = *((DIGIT "-" DIGIT ) / digitMapLetter)
digitMapLetter       = DIGIT / %x41-4B / %x61-6B / ; 0-9, A-K, (a-k)
                       "L"                  ; Long duration modifier

;at-most-once
auditItem            = ( MuxToken / ModemToken / MediaToken /
                         SignalsToken / EventBufferToken /
                         DigitMapToken / StatsToken / EventsToken /
```



```
                         ObservedEventsToken / PackagesToken )

serviceChangeDescriptor = ServicesToken LBRKT serviceChangeParm
                          *(COMMA serviceChangeParm) RBRKT

serviceChangeParm    = (serviceChangeMethod / serviceChangeReason /
                        serviceChangeDelay / serviceChangeAddress /
                        serviceChangeProfile / extension / TimeStamp /
                        serviceChangeMgcId / serviceChangeVersion )

serviceChangeReplyDescriptor = ServicesToken LBRKT
                       servChgReplyParm *(COMMA servChgReplyParm) RBRKT

;at-most-once. Version is REQUIRED on first ServiceChange response
servChgReplyParm     = (serviceChangeAddress / serviceChangeMgcId /
                        serviceChangeProfile / serviceChangeVersion )
serviceChangeMethod  = MethodToken EQUAL (FailoverToken /
                        ForcedToken / GracefulToken / RestartToken /
                        DisconnectedToken / HandOffToken /
                        extensionParameter)

serviceChangeReason  = ReasonToken  EQUAL VALUE
serviceChangeDelay   = DelayToken   EQUAL UINT32
serviceChangeAddress = ServiceChangeAddressToken EQUAL VALUE
serviceChangeMgcId   = MgcIdToken   EQUAL mId
serviceChangeProfile = ProfileToken EQUAL NAME SLASH Version
serviceChangeVersion = VersionToken EQUAL Version
extension            = extensionParameter parmValue

packagesDescriptor   = PackagesToken LBRKT packagesItem
                       *(COMMA packagesItem) RBRKT

Version              = 1*2(DIGIT)
packagesItem         = NAME "-" UINT16

TimeStamp            = Date "T" Time ; per ISO 8601:1988
; Date = yyyymmdd
Date                 = 8(DIGIT)
; Time = hhmmssss
```

```
Time                  = 8(DIGIT)
statisticsDescriptor = StatsToken LBRKT statisticsParameter
                       *(COMMA statisticsParameter ) RBRKT

;at-most-once per item
statisticsParameter   = pkgdName EQUAL VALUE

topologyDescriptor    = TopologyToken LBRKT terminationA COMMA
                        terminationB COMMA topologyDirection RBRKT
```



```
terminationA          = TerminationID
terminationB          = TerminationID
topologyDirection     = BothwayToken / IsolateToken / OnewayToken

priority              = PriorityToken EQUAL UINT16

extensionParameter    = "X"  ("-" / "+") 1*6(ALPHA / DIGIT)

; octetString is used to describe SDP defined in RFC2327.
; Caution should be taken if CRLF in RFC2327 is used.
; To be safe, use EOL in this ABNF.
; Whenever "}" appears in SDP, it is escaped by "
octetString           = *(nonEscapeChar)
nonEscapeChar         = ( "" / %x01-7C / %x7E-FF )
quotedString          = DQUOTE 1* (SafeChar / RestChar/ WSP) DQUOTE

UINT16                = 1*5(DIGIT)  ; %x0-FFFF
UINT32                = 1*10(DIGIT) ; %x0-FFFFFFFF

NAME                  = ALPHA *63(ALPHA / DIGIT / "_" )
VALUE                 = quotedString / 1* (SafeChar)
SafeChar              = DIGIT / ALPHA / "+" / "-" / "&" /
                        "!" / "_" / "/" / "'" / "?" / "@" /
                        "^" / "`" / "~" / "*" / "$" / "
                        "(" / ")" / "%" / "|" / "."

EQUAL                 = LWSP %x3D LWSP ; "="
COLON                 = %x3A           ; ":"
LBRKT                 = LWSP %x7B LWSP ; "{"
RBRKT                 = LWSP %x7D LWSP ; "}"
COMMA                 = LWSP %x2C LWSP ; ","
DOT                   = %x2E           ; "."
SLASH                 = %x2F           ; "/"
ALPHA                 = %x41-5A / %x61-7A ; A-Z / a-z
DIGIT                 = %x30-39          ; 0-9
DQUOTE                = %x22             ; " (Double Quote)
HEXDIG                = ( DIGIT / "A" / "B" / "C" / "D" / "E" / "F" )
SP                    = %x20        ; space
HTAB                  = %x09        ; horizontal tab
CR                    = %x0D        ; Carriage return
LF                    = %x0A        ; linefeed
LWSP                  = *( WSP / COMMENT / EOL )
```

```
EOL                 = (CR [LF] / LF )
WSP                 = SP / HTAB ; white space
SEP                 = ( WSP / EOL / COMMENT) LWSP
COMMENT             = ";" *(SafeChar/ RestChar / WSP / %x22) EOL
RestChar            = ";" / "[" / "]" / "{" / "}" / ":" / "," / "#" /
                      "<" / ">" / "="
```



```
AddToken            = ("Add"                / "A")
AuditToken          = ("Audit"              / "AT")
AuditCapToken       = ("AuditCapability"    / "AC")
AuditValueToken     = ("AuditValue"         / "AV")
AuthToken           = ("Authentication"     / "AU")
BothwayToken        = ("Bothway"            / "BW")
BriefToken          = ("Brief"              / "BR")
BufferToken         = ("Buffer"             / "BF")
CtxToken            = ("Context"            / "C")
ContextAuditToken   = ("ContextAudit"       / "CA")
DigitMapToken       = ("DigitMap"           / "DM")
DiscardToken        = ("Discard"            / "DS")
DisconnectedToken   = ("Disconnected"       / "DC")
DelayToken          = ("Delay"              / "DL")
DurationToken       = ("Duration"           / "DR")
EmbedToken          = ("Embed"              / "EB")
EmergencyToken      = ("Emergency"          / "EM")
ErrorToken          = ("Error"              / "ER")
EventBufferToken    = ("EventBuffer"        / "EB")
EventsToken         = ("Events"             / "E")
FailoverToken       = ("Failover"           / "FL")
ForcedToken         = ("Forced"             / "FO")
GracefulToken       = ("Graceful"           / "GR")
H221Token           = ("H221" )
H223Token           = ("H223" )
H226Token           = ("H226" )
HandOffToken        = ("HandOff"            / "HO")
InactiveToken       = ("Inactive"           / "IN")
IsolateToken        = ("Isolate"            / "IS")
InSvcToken          = ("InService"          / "IV")
KeepActiveToken     = ("KeepActive"         / "KA")
LocalToken          = ("Local"              / "L")
LocalControlToken   = ("LocalControl"       / "O")
LoopbackToken       = ("Loopback"           / "LB")
MediaToken          = ("Media"              / "M")
MegacopToken        = ("MEGACO"             / "!")
MethodToken         = ("Method"             / "MT")
MgcIdToken          = ("MgcIdToTry"         / "MG")
ModeToken           = ("Mode"               / "MO")
ModifyToken         = ("Modify"             / "MF")
ModemToken          = ("Modem"              / "MD")
MoveToken           = ("Move"               / "MV")
MTPToken            = ("MTP")
MuxToken            = ("Mux"                / "MX")
NotifyToken         = ("Notify"             / "N")
```

```
NotifyCompletionToken      = ("NotifyCompletion"      / "NC")
ObservedEventsToken        = ("ObservedEvents"        / "OE")
OnewayToken                = ("Oneway"                / "OW")
```



```
OnOffToken                 = ("OnOff"                 / "OO")
OutOfSvcToken              = ("OutOfService"          / "OS")
PackagesToken              = ("Packages"              / "PG")
PendingToken               = ("Pending"               / "PN")
PriorityToken              = ("Priority"              / "PR")
ProfileToken               = ("Profile"               / "PF")
ReasonToken                = ("Reason"                / "RE")
RecvonlyToken              = ("ReceiveOnly"           / "RC")
ReplyToken                 = ("Reply"                 / "P")
RestartToken               = ("Restart"               / "RS")
RemoteToken                = ("Remote"                / "R")
ReservedToken              = ("Reserved"                        / "RV")
SendonlyToken              = ("SendOnly"              / "SO")
SendrecvToken              = ("SendReceive"           / "SR")
ServicesToken              = ("Services"              / "SV")
ServiceStatesToken         = ("ServiceStates"         / "SI")
ServiceChangeToken         = ("ServiceChange"         / "SC")
ServiceChangeAddressToken  = ("ServiceChangeAddress"  / "AD")
SignalListToken            = ("SignalList"            / "SL")
SignalsToken               = ("Signals"               / "SG")
SignalTypeToken            = ("SignalType"            / "SY")
StatsToken                 = ("Statistics"            / "SA")
StreamToken                = ("Stream"                / "ST")
SubtractToken              = ("Subtract"              / "S")
SynchISDNToken             = ("SynchISDN"             / "SN")
TerminationStateToken      = ("TerminationState"      / "TS")
TestToken                  = ("Test"                  / "TE")
TimeOutToken               = ("TimeOut"               / "TO")
TopologyToken              = ("Topology"              / "TP")
TransToken                 = ("Transaction"           / "T")
V18Token                   = ("V18")
V22Token                   = ("V22")
V22bisToken                = ("V22b")
V32Token                   = ("V32")
V32bisToken                = ("V32b")
V34Token                   = ("V34")
V76Token                   = ("V76")
V90Token                   = ("V90")
V91Token                   = ("V91")
```

ANNEX C TAGS FOR MEDIA STREAM PROPERTIES (NORMATIVE)

Parameters for Local descriptors and Remote descriptors are specified as tag-value pairs if binary encoding is used for the protocol. This annex contains the property names (PropertyID), the tags (Property Tag), type of the property (Type) and the values (Value).Values presented in the

Value field when the field contains references shall be regarded as "information". The reference contains the normative values. If a value field does not contain a reference then the values in that field can be considered as "normative".

Tags are given as hexadecimal numbers in this annex. When setting the value of a property, a MGC may underspecify the value according to one of the mechanisms specified in section 7.1.1.

For type "enumeration" the value is represented by the value in brackets, e.g., Send(0), Receive(1).

C.1. General Media Attributes

PropertyID	Tag	Type	Value
Media	1001	Enumeration	Audio(0), Video(1) ,Data(2),
TransMode	1002	Enumeration	Send(0), Receive(1), Send&Receive(2)
NumChan	1003	UINT	0-255
SamplingRate	1004	UINT	0-2^32
Bitrate	1005	Integer	(0..4294967295) Note-units of 100 bit
Acodec	1006		. Audio Codec Type
Samplepp	1007	UINT	Maximum samples per packet:0-65535
Silencesupp	1008	BOOLEAN	Silence Suppression
Encrypttype	1009	Octet str	Ref.: rec. H.245
Encryptkey	100A	Octet str	SIZE(0..65535) Encryption key
Echocanc	100B	Enumeration	Echo Canceller:Off(0),G.165(1),G168(2)
Gain	100C	UINT	Gain in db: 0-65535
Jitterbuff	100D	UINT	Jitter buffer size in ms: 0-65535
PropDelay	100E	UINT	Propagation Delay: 0..65535
RTPpayload	100F	integer	Payload type in RTP Profile

C.2. Mux Properties

PropertyID	Tag	Type	Value
H.221	2001	Octet string	H222LogicalChannelParameters
H223	2002	Octet string	H223LogicalChannelParameters
V76	2003	Octet String	V76LogicalChannelParameters
H2250	2004	Octet String	H2250LogicalChannelParameters

C.3. General bearer properties

PropertyID	Tag	Type	Value
Mediatx	3001	Enumeration	Media Transport Type
BIR	3002	4 OCTET	Value depends on transport
NSAP	3003	20 OCTET	Ref: ITU X.213 Annex A

C.4. General ATM properties

PropertyID	Tag	Type	Value
AESA	4001	20 OCTETS	ATM End System Address
VPVC	4002	2x16b int	VPC-VCI
SC	4003	4 bits	Service Category
BCOB	4004	5b integer	Broadband Bearer Class
BBTC	4005	octet	Broadband Transfer Capability
ATC	4006	Enumeration	I.371 ATM Traffic Cap.
STC	4007	2 bits	Susceptibility to clipping
UPCC	4008	2 bits	User Plane Connection config
PCR0	4009	24b integer	Peak Cell Rate CLP=0
SCR0	400A	24b integer	Sustainable Cell Rate CLP=0
MBS0	400B	24b integer	Maximum Burst Size CLP=0
PCR1	400C	24b integer	Peak Cell Rate CLP=0+1
SCR2	400D	24b integer	Sustain. Cell Rate CLP=0+1
MBS3	400E	24b integer	Maximum Burst Size CLP=0+1
BEI	400F	Boolean	Best Effort Indicator
TI	4010	Boolean	Tagging
FD	4011	Boolean	Frame Discard
FCDV	4012	24b integer	Forward P-P CDV
BCDV	4013	24b integer	Backward P-P CDV
FCLR0	4014	8b integer	Fwd Cell Loss Ratio CLP=0
BCLR0	4015	8b integer	Bkwd P-P CLR CLP=0
FCLR1	4016	8b integer	Fwd Cell Loss Ratio CLP=0+1
BCLR1	4017	8b integer	Bkwd P-P CLR CLP=0+1
FCDV	4018	24b integer	Fwd Cell Delay Variation
BCDV	4019	24b integer	Bkwd Cell Delay Variation
FACDV	401A	24b integer	Fwd Acceptable P-P-P CDV
BACDV	401B	24b integer	Bkwd Acceptable P-P CDV
FCCDV	401C	24b integer	Fwd Cumulative P-P CDV
BCCDV	401D	24b integer	Bkwd Cumulative P-P CDV
FCLR	401E	8b integer	Acceptable Fwd CLR
BCLR	401F	8b integer	Acceptable Bkwd CLR
EETD	4020	16b integer	End-to-end transit delay
Mediatx	4021		AAL Type
QosClass	4022	Integer	0-4 Qos Class
AALtype	4023	1 OCTET	AAL Type Reference

C.5. Frame Relay

PropertyID	Tag	Type	Value
DLCI	5001	Unsigned Integer	Data link connection id
CID	5002	Unsigned Integer	sub-channel id.
SID	5003	Unsigned Integer	silence insertion descriptor
Payload	5004	Unsigned Integer	Primary Payload Type

C.6. IP

PropertyID	Tag	Type	Value
IPv4	6001	32 BITS	Ipv4Address or Ipv6 address
IPv6	6002	128 BITS	IPv6 Address
Port	6002	0-65535	Port
TCP	6003	Boolean	
UDP	6004	Boolean	

C.7. ATM AAL2

PropertyID	Tag	Type	Value
AESA	7001	20 OCTETS	AAL2 service endpoint address
BIR	See C.3	4 OCTETS	Served user generated reference
ALC	7002	12 OCTETS	AAL2 link
SSCS	7003	8..14 OCTETS	Service specific convergence sublayer
SUT	7004	1..254 octets	Served user transport param
TCI	7005	BOOLEAN	Test connection
Timer_CU	7006	32b integer	Timer-CU
MaxCPSSDU	7007	8b integer	Max. Common Part Sublayer SDU
SCLP	7008	Boolean	Set Cell Local PriorityLP bit
EETR	7009	Boolean	End to End Timing Required

C.8. ATM AAL1

PropertyID	Tag	Type	Value
BIR	See Table C.3	4 OCTETS	GIT (Generic Identifier Transport)
AAL1ST	8001	1 OCTET	AAL1 Subtype
CBRR	8002	1 OCTET	CBR Rate
SCRI	8003	1 OCTET	Src. Clock Freq. Recovery Method
ECM	8004	1 OCTET	Error Correction Method
SDTB	8005	16b integer	Structured Data Transfer Blksize
PFCI	8006	8b integer	Partially filled cells identifier
EETR	See Table C.7	See Table C.7	

C.9. Bearer Capabilities

PropertyID	Tag	Type	Value
TMR	9001	1 OCTET	Transmission Medium Requirement
TMRSR	9002	1 OCTET	Trans. Medium Requirement Subrate

Contcheck	0003	BOOLEAN	Continuity Check
ITC	9004	5 BITS	Information Transfer Capability
TransMode	9005	2 BITS	Transfer Mode
TransRate	9006	5 BITS	Transfer Rate
MULT	9007	7 BITS	Rate Multiplier
USI	9008	5 BITS	User Information Layer 1 Protocol
Syncasync	9009	BOOLEAN	Synchronous-Asynchronous
Userrate	900B	5 BITS	User Rate Reference
INTRATE	900C	2 BITS	Intermediate Rate
Nictx	900D	BOOLEAN	Tx Network Independent Clock
Nicrx	900E	BOOLEAN	Rx Network independent clock
Flowconttx	900F	BOOLEAN	Tx Flow Control
Flowcontrx	9010	BOOLEAN	Rx Flow control
Rateadapthdr	9011	BOOLEAN	Rate adapt header-no header
Multiframe	9012	BOOLEAN	Multiple frame estab.
OPMODE	9013	BOOLEAN	Mode of operation
Llidnegot	9014	BOOLEAN	Logical link identifier neg.
Assign	9015	BOOLEAN	Assignor-assignee
Inbandneg	9016	BOOLEAN	In-band or out-band negotiation
Stopbits	9017	2 BITS	Number of stop bits
Databits	9018	2 BIT	Number of data bits
Parity	9019	3 BIT	Parity information
Duplexmode	901A	BOOLEAN	Mode duplex
Modem	901B	6 BIT	Modem Type
layer2prot	901C	5 BIT	User info layer 2 protocol
layer3prot	901D	5 BIT	User info layer 3 protocol
addlayer3prot	901E	OCTET	Addl User Info L3 protocol
DialledN	901F	10 OCTETS	Dialled Number
DiallingN	9020	10 OCTETS	Dialling Number
ECHOCI	9021	Enumeration	Echo Control Information
NCI	9022	1 OCTET	Nature of Connection Indicators

C.10. AAL5 Properties

PropertyID	Tag	Type	Value
FMSDU	A001	32b integer	Forward Maximum CPCS-SDU Size:
BMSDU	A002	2b integer	Backwards Maximum CPCS-SDU Size
SSCS	See C.7	See C.7	See table C.
SC	See C.4	See C.4	See table C.4

C.11. SDP Equivalents

PropertyID	Tag	Type	Value
SDP_V	B001	STRING	Protocol Version
SDP_O	B002	STRING	Owner-creator and session ID
SDP_S	B003	STRING	Sesson name

SDP_I	B004	STRING	Session identifier
SDP_U	B005	STRING	URI of descriptor
SDC_E	B006	STRING	email address
SDP_P	B007	STRING	phone number
SDP_C	B008	STRING	Connection information
SDP_B	B009	STRING	Bandwidth Information
SDP_Z	B00A	STRING	time zone adjustment
SDP_K	B00B	STRING	Encryption Key
SDP_A	B00C	STRING	Zero or more session attributes
SDP_T	B00D	STRING	Active Session Time
SDP_R	B00E	STRING	Zero or more repeat times

C.12. H.245

OLC	C001	octet string	H.245 OpenLogicalChannel structure.
OLCack	C002	octet string	H.245 OpenLogicalChannelAck structure.
OLCcnf	C003	octet string	OpenLogicalChannelConfirm structure.
OLCrej	C004	octet string	OpenLogicalChannelReject structure.
CLC	C005	octet string	CloseLogicalChannel structure.
CLCack	C006	octet string	CloseLogicalChannelAck structure.

ANNEX D TRANSPORT OVER IP (NORMATIVE)

D.1. Transport over IP/UDP using Application Level Framing

Protocol messages defined in this document may be transmitted over UDP. When no port is provided by the peer (see section 7.2.8), the commands should be sent to the default number, xxxx for text-encoded operation or yyyy for binary-encoded operation. Responses must always be sent to the address and port from which the corresponding commands were sent.

D.1.1. Providing At-Most-Once Functionality

Messages, being carried over UDP, may be subject to losses. In the absence of a timely response, commands are repeated. Most commands are

not idempotent. The state of the MG would become unpredictable if, for example, Add commands were executed several times. The transmission procedures shall thus provide an "At- Most-Once" functionality.

Peer protocol entities are expected to keep in memory a list of the responses that they sent to recent transactions and a list of the transactions that are currently outstanding. The transaction identifiers of incoming messages are compared to the transaction identifiers of the recent responses to the same MId. If a match is found, the entity does not execute the transaction, but simply repeats the response. The

remaining messages will be compared to the list of current transactions. If a match is found, indicating a duplicate transaction, the entity does not execute the transaction, which is simply ignored.

The procedure uses a long timer value, noted LONG-TIMER in the following. The timer should be set larger than the maximum duration of a transaction, which should take into account the maximum number of repetitions, the maximum value of the repetition timer and the maximum propagation delay of a packet in the network. A suggested value is 30 seconds.

The copy of the responses may be destroyed either LONG-TIMER seconds after the response is issued, or when the entity receives a confirmation that the response has been received, through the "Response Acknowledgement parameter". For transactions that are acknowledged through this parameter, the entity shall keep a copy of the transaction-id for LONG-TIMER seconds after the response is issued, in order to detect and ignore duplicate copies of the transaction request that could be produced by the network.

D.1.2. Transaction identifiers and three-way handshake

Transaction identifiers are 32 bit integer numbers. A Media Gateway Controller may decide to use a specific number space for each of the MGs that they manage, or to use the same number space for all MGs that belong to some arbitrary group. MGCs may decide to share the load of managing a large MG between several independent processes. These processes will share the same transaction number space. There are multiple possible implementations of this sharing, such as having a centralized allocation of transaction identifiers, or pre-allocating non-overlapping ranges of identifiers to different processes. The implementations shall guarantee that unique transaction identifiers are allocated to all transactions that originate from a logical MGC (identical mId). MGs can simply detect duplicate transactions by looking at the transaction identifier and mId only.

The Response Acknowledgement parameter can be found in any message. It carries a set of "confirmed transaction-id ranges". Entities may choose

to delete the copies of the responses to transactions whose id is included in "confirmed transaction-id ranges" received in the transaction response messages. They should silently discard further commands when the transaction-id falls within these ranges.

The "confirmed transaction-id ranges" values shall not be used if more than LONG-TIMER seconds have elapsed since the MG issued its last response to that MGC, or when a MG resumes operation. In this situation, transactions should be accepted and processed, without any test on the transaction-id.

Messages that carry the "Response Acknowledgement" parameter may be transmitted in any order. The entity shall retain the "confirmed

transaction-id ranges" received in for LONG- TIMER seconds.

The ASN.1 of Annex A is modified as follows. The definition of Transaction of Annex A is replaced by

```
Transaction ::= CHOICE
{
        transactionRequest      TransactionRequest,
        transactionPending      TransactionPending,
        transactionReply                TransactionReply,
        transactionResponseAck  TransactionResponseAck
}
```

The definition of TransactionResponseAck reads

```
TransactionResponseAck ::= SEQUENCE
{
        firstAck        TransactionId,
        lastAck TransactionId OPTIONAL
}
```

If only the firstAck is present in a response acknowledgement, only one transaction is acknowledged. If both firstAck and lastAck are present, then the range of transactions from firstAck to lastAck is acknowledged.

The ABNF of Annex B is modified so that:

```
transactionList      = 1*(transactionRequest / transactionReply /
                       transactionPending / transactionResponseAck)

transactionResponseAck = ResponseAckToken LBRKT transactionAck
                                    *(COMMA transactionAck) RBRKT
```



```
transactionAck = transactionID / (transactionID "-" transactionID)
ResponseAckToken = "TransactionResponseAck" | "K"
```

D.1.3. Computing retransmission timers

It is the responsibility of the requesting entity to provide suitable time outs for all outstanding transactions, and to retry transactions when time outs have been exceeded. Furthermore, when repeated transactions fail to be acknowledged, it is the responsibility of the requesting entity to seek redundant services and/or clear existing or pending connections.

The specification purposely avoids specifying any value for the retransmission timers. These values are typically network dependent. The retransmission timers should normally estimate the timer value by

measuring the time spent between the sending of a command and the return of a response. One possibility is to use the algorithm implemented in TCP-IP, which uses two variables:

* The average acknowledgement delay, AAD, estimated through an exponentially smoothed average of the observed delays.

* The average deviation, ADEV, estimated through an exponentially smoothed average of the absolute value of the difference between the observed delay and the current average.

The retransmission timer, in TCP, is set to the sum of the average delay plus N times the average deviation. The maximum value of the timer should however be bounded for the protocol defined in this document, in order to guarantee that no repeated packet would be received by the gateways after LONG-TIMER seconds. A suggested maximum value is 4 seconds. After any retransmission, the entity should do the following:

* It should double the estimated value of the average delay, AAD

* It should compute a random value, uniformly distributed between 0.5 AAD and AAD

* It should set the retransmission timer to the sum of that random value and N times the average deviation.

This procedure has two effects. Because it includes an exponentially increasing component, it will automatically slow down the stream of messages in case of congestion. Because it includes a random component, it will break the potential synchronization between notifications triggered by the same external event.

D.1.4. Provisional responses

Executing some transactions may require a long time. Long execution times may interact with the timer based retransmission procedure. This may result either in an inordinate number of retransmissions, or in timer values that become too long to be efficient. Entities that can predict that a transaction will require a long execution time may send a provisional response, "Transaction Pending". They should send this response if they receive a repetition of a transaction that is still being executed.

Entities that receive a Transaction Pending shall switch to a different repetition timer for repeating requests. The root termination has a property (ProvisionalResponseTimerValue), which can be set to the requested maximum number of milliseconds between receipt of a command and transmission of the TransactionPending response. Upon receipt of a final response, an immediate confirmation shall be sent, and normal repetition timers shall be used thereafter. Receipt of a Transaction Pending after receipt of a reply shall be ignored.

D.1.5. Repeating Requests, Responses and Acknowledgements

The protocol is organized as a set of transactions, each of which is composed request and a response, commonly referred to as an acknowledgement. The protocol messages, being carried over UDP, may be subject to losses. In the absence of a timely response, transactions are repeated. Entities are expected to keep in memory a list of the responses that they sent to recent transactions, i.e. a list of all the responses they sent over the last LONG-TIMER seconds, and a list of the transactions that are currently being executed.

The repetition mechanism is used to guard against three types of possible errors:

* transmission errors, when for example a packet is lost due to noise on a line or congestion in a queue;

* component failure, when for example an interface to a entity becomes unavailable;

* entity failure, when for example an entire entity become unavailable.

The entities should be able to derive from the past history an estimate of the packet loss rate due to transmission errors. In a properly configured system, this loss rate should be kept very low, typically less than 1%. If a Media Gateway Controller or a Media Gateway has to repeat a message more than a few times, it is very legitimate to assume that

something else than a transmission error is occurring. For example, given a loss rate of 1%, the probability that five consecutive transmission attempts fail is 1 in 100 billion, an event that should occur less than once every 10 days for a Media Gateway Controller that processes 1 000 transactions per second. (Indeed, the number of repetition that is considered excessive should be a function of the prevailing packet loss rate.) We should note that the "suspicion threshold", which we will call "Max1", is normally lower than the "disconnection threshold", which should be set to a larger value.

A classic retransmission algorithm would simply count the number of successive repetitions, and conclude that the association is broken after retransmitting the packet an excessive number of times (typically between 7 and 11 times.) In order to account for the possibility of an undetected or in-progress "failover", we modify the classic algorithm so that if the Media Gateway receives a valid ServiceChange message announcing a failover, it will start transmitting outstanding commands to that new MGC. Responses to commands are still transmitted to the source address of the command.

In order to automatically adapt to network load, this document specifies exponentially increasing timers. If the initial timer is set to 200

milliseconds, the loss of a fifth retransmission will be detected after about 6 seconds. This is probably an acceptable waiting delay to detect a failover. The repetitions should continue after that delay not only in order to perhaps overcome a transient connectivity problem, but also in order to allow some more time for the execution of a failover - waiting a total delay of 30 seconds is probably acceptable.

It is, however, important that the maximum delay of retransmissions be bounded. Prior to any retransmission, it is checked that the time elapsed since the sending of the initial datagram is no greater than T-MAX. If more than T-MAX time has elapsed, the MG concludes that the MGC has failed, and it begins its recovery process. When the MG establishes a new control association, it can retransmit to the new MGC. The value T-MAX is related to the LONG- TIMER value: the LONG-TIMER value is obtained by adding to T-MAX the maximum propagation delay in the network.

D.2. Using TCP

Protocol messages as defined in this document may be transmitted over TCP. When no port is specified by the other side (see section 7.2.8), the commands should be sent to the default port. The defined protocol has messages as the unit of transfer, while TCP is a stream-oriented protocol. TPKT, according to RFC1006 SHALL be used to delineate messages within the TCP stream.

In a transaction-oriented protocol, there are still ways for transaction requests or responses to be lost. As such, it is recommended that entities using TCP transport implement application level timers for each request and each response, similar to those specified for application level framing over UDP.

D.2.1. Providing the At-Most-Once functionality

Messages, being carried over TCP, are not subject to transport losses, but loss of a transaction request or its reply may nonetheless be noted in real implementations. In the absence of a timely response, commands are repeated. Most commands are not idempotent. The state of the MG would become unpredictable if, for example, Add commands were executed several times.

To guard against such losses, it is recommended that entities follow the procedures in section D.1.1

D.2.2. Transaction identifiers and three way handshake

For the same reasons, it is possible that transaction replies may be lost even with a reliable delivery protocol such as TCP. It is recommended that entities follow the procedures in section D.1.2

D.2.3. Computing retransmission timers

With reliable delivery, the incidence of loss of a transaction request or reply is expected to be very low. Therefore, only simple timer mechanisms are required. Exponential back-off algorithms should not be necessary, although they could be employed where, as in an MGC, the code to do so is already required, since MGCs must implement ALF/UDP as well as TCP.

D.2.4. Provisional responses

As with UDP, executing some transactions may require a long time. Entities that can predict that a transaction will require a long execution time may send a provisional response, "Transaction Pending". They should send this response if they receive a repetition of a transaction that is still being executed.

Entities that receive a Transaction Pending shall switch to a longer repetition timer for that transaction.

Entities shall retain Transactions and replies until they are confirmed. The basic procedure of section D.1.4 should be followed, but simple timer values should be sufficient.

D.2.5. Ordering of commands

TCP provides ordered delivery of transactions. No special procedures are required. It should be noted that ALF/UDP allows sending entity to modify its behavior under congestion, and in particular, could reorder transactions when congestion is encountered. TCP could not achieve the same results.

ANNEX E BASIC PACKAGES

This Annex contains definitions of some packages for use with MEGACO.

E.1. Generic

PackageID: g (0x000e)
Version: 1
Extends: None
Description:
Generic package for commonly encountered items

E.1.1. Properties

None

E.1.2. Events

Cause

```
      EventID:  cause (0x0001)
      Generic error event
        Event Descriptor Parameters:
              General Cause
                  ParameterID: Generalcause (0x0001)
                  This parameter groups the failures into six groups,
                  which the MGC may act upon.
                  Possible values:      Enumerated,
                        "NR" Normal Release (0x0001)
                              "UR" Unavailable Resources (0x0002)
                              "FT" Failure, Temporary (0x0003)
                              "FP" Failure, Permanent (0x0004)
                              "IW" Interworking Error (0x0005)
                              "UN" Unsupported (0x0006)
             Failure Cause
                        ParameterID: Failurecause (0x0002)
                  Possible Values:      OCTET STRING
                  Description: The Release Cause is the value generated
                       by the Released equipment, i.e. a released network
                       connection.  The concerned value is defined in the
```



```
                       appropriate bearer control protocol.
Signal Completion
        EventID: sc (0x0002)
        Indicates termination of one or more signals for which the
        notifyCompletion parameter was set to "ON".  For further
        procedural description, see sections 7.1.11, 7.1.17, and 7.2.7.

        ObservedEvents Descriptor parameters:
                Signal Identity
                        ParameterID:  SigID (0x0001)
                        This parameter identifies the signals which have terminated.
                        Type: list
                        Possible values: a list of signals and/or sequential
                        signal lists which have terminated.  A signal outside
                        of a sequential signal list shall be identified using
                        the pkgdName syntax without wildcarding.  An
                        individual signal inside of a sequential signal list
                        shall be identified using the sequential signal list
                        syntax with the correct signal list     identifier,
                        enclosing the name of the specific signal which
                        terminated in pkgdName syntax.

                Termination Method
                        ParameterID:  Meth (0x0002)
                        Indicates the means by which the signal terminated.
                        Type: enumeration
                        Possible values:
                                "TO" (0x0001) Duration expired
                                "EV" (0x0002) Interrupted by event
                                "SD" (0x0003) Halted by new Signals Descriptor
                                "FL" (0x0004) Failure
```

19.1.3. Signals

None

19.1.4. Statistics

None

19.2. Base Root Package

Base Root Package

PackageID: root (0x000f)
Version: 1
Extends: None
Description:
This package defines Gateway wide properties.

19.2.1. Properties

MaxNrOfContexts
PropertyID: maxNumberOfContexts (0x0001)
The value of this property gives the maximum number of contexts that can exist at any time. The NULL context is not included in this number.
Type: Double
Possible values: 1 and up
MaxTerminationsPerContext
PropertyID: maxTerminationsPerContext (0x0002)
The maximum number of allowed terminations in a context, see section 6.1
Type: Integer

```
            Possible Values: any integer
            Defined In: TerminationState
      normalMGExecutionTime
            PropertyId: normalMGExecutionTime (0x0003)
            Settable by the MGC to indicate the interval within which
                  the MGC expects a response to any transaction from
                  the MG (exclusive of network delay)
            Type: Integer
            Possible Values: any integer, represents milliseconds
      normalMGCExecutionTime
            PropertyId: normalMGCExecutionTime (0x0004)
            Settable by the MGC to indicate the interval within which
                  the MG should expects a response to any transaction
                  from the MGC (exclusive of network delay)


Cuervo, Hill, Greene, Huitema, Rayhan, Rosen, Segers          [Page 116]


Internet draft              MEGACO Protocol             February 8, 2000


            Type: Integer
            Possible Values: any integer, represents milliseconds
      ProvisionalResponseTimerValue
            PropertyId: ProvisionalResponseTimerValue (0x0005)
            Indicates the time within which to expect a Pending
                  Response if a Transaction cannot be completed.
                  Initially set to normalMGExecutionTime or
                  normalMGCExecutionTime as appropriate, plus network
                  delay, but may be lowered.
      Pattern
            PropertyId: Pattern (0x0006)
            Name pattern of a set of terminations in the gateway.
                  Used to discover the names of terminations that
                  can be audited.  Includes ephemeral terminations.
                  MGs SHOULD use one pattern for each type of
                  termination (same packages implemented), but no
                  two Patterns can have the same value.
            Type: String
            Possible Value:
                  A string of up to 64 characters using the following
                  characters:
                        a-z,A-Z,0-9, and "/" - the actual character
                            in the name
                        * - any set of characters
                        ?a - any single character
                        ?0 - any digit
                        ?a - any alpha
                        [n,n,..,n] - alternatives, one of the
                            alternatives listed, n can be a substring
                            of alphas or digits
                        [n-n] - range, any number in the range,
                            n can be a number or an alpha, for example
                              [00-27] or [a-e]
                    Note, mixing of alternatives or ranges is allowed,
                        as in: [0,3-5,8]
            Characteristics: Read-only
      MaxPatterns
```

PropertyId: MaxPatterns (0x0007)
The number of patterns in the gateway
Type: Integer
Possible Value: any integer
Characteristics: Read-only

PatternNum
PropertyId: PatternNum (0x0008)
Which pattern to read, zero based. Set by the MGC to
read a specific pattern in Pattern
Type: Integer
Possible Value: any integer less than MaxPatterns

E.2.2. Events

None

E.2.3. Signals

None

E.2.4. Statistics

None

E.2.5. Procedures

None

E.3. Tone Generator Package

PackageID: tonegen (0x0001)
Version: 1
Extends: None
Description:
This package defines signals to generate audio tones.
This package does not specify parameter values. It is
intended to be extendable. Generally, tones are defined
as an individual signal with a parameter, ind,
representing "interdigit" time delay, and a tone id to
be used with playtones. A tone id should be kept
consistent with any tone generation for the same tone.
MGs are expected to be provisioned with the characteristics
of appropriate tones for the country in which the MG is located.

E.3.1. Properties

None

E.3.2. Events

None

E.3.3. Signals

Play tone

SignalID: pt (0x0001)
Plays audio tone over an audio channel
Signal Type: Brief

Duration: Provisioned
Additional Parameters:
Tone id list
ParameterID: tl (0x0001)
Type: list of tone ids.
List of tones to be played in sequence.
The list SHALL contain one or more tone ids.
Inter signal duration
ParameterID: ind (0x0002)
Type: integer.
Timeout between two consecutive tones in milliseconds

No tone ids are specified in this package. Packages that extend this package can add possible values for tone id as well as adding individual tone signals

E.3.4. Statistics

None

E.3.5. Procedures

None

E.4. Tone Detection Package

PackageID: tonedet (0x0002)
Version: 1
Extends: None
This Package defines events for audio tone detection.
Tones are selected by name (tone id). MGs are expected to be provisioned with the characteristics of appropriate tones for the country in which the MG is located.

This package does not specify parameter values. It is intended to be extendable.

E.4.1. Properties

None

E.4.2. Events

Start tone detected
EventID: std, 0x0001
Detects the start of a tone. The characteristics of positive

tone detection is implementation dependent.
EventsDescriptor parameters:
Tone id list
ParameterID: tl (0x0001)
Type: list of tone ids
Possible values: The only tone id defined in this package is "wild card" which is "*" in text encoding and 0x0000 in binary. Extensions to this package would add possible values for tone id. If tl is "wild card", any tone id is detected
ObservedEventsDescriptor parameters:
Tone id
ParameterID: tid (0x0003)
Type: Enumeration
Possible values: "wildcard" as defined above is the only value defined in this package. Extensions to this package would add additional possible values for tone id

End tone detected
EventID: etd, 0x0002
Detects the end of a tone.
EventDescriptor parameters:
Tone id list
ParameterID: tl (0x0001)
Type: enumeration or list of enumerated types
Possible values: No possible values are specified in this package. Extensions to this package would add possible values for tone id
ObservedEventsDescriptor parameters:
Tone id
ParameterID: tid (0x0003)
Type: Enumeration
Possible values: "wildcard" as defined above is the only value defined in this package. Extensions to this package would add possible values for tone id
Duration
ParameterId: dur (0x0002)
Type: integer, in milliseconds
This parameter contains the duration of the tone from first detection until it stopped.

Long tone detected
EventID: ltd, 0x0003
Detects that a tone has been playing for at least a certain amount of time

```
      EventDescriptor parameters:
            Tone id list


Cuervo, Hill, Greene, Huitema, Rayhan, Rosen, Segers          [Page 120]


Internet draft              MEGACO Protocol             February 8, 2000


                  ParameterID: tl (0x0001)
                  Type: enumeration or list
                  Possible values: "wildcard" as defined above is the
                        only value defined in this package. Extensions
                        to this package would add possible values for
                        tone id
            Duration:
                  ParameterID: dur (0x0002)
                  Type: integer, duration to test against
                  Possible values: any legal integer, expressed in
                        milliseconds
      ObservedEventsDescriptor parameters:
            Tone id:
                  ParameterID: tid (0x0003)
                  Possible values: No possible values are specified
                  in this package.  Extensions to this package
                  would add possible values for tone id
```

E.4.3. Signals

None

E.4.4. Statistics

None

E.4.5. Procedures

None

E.5. Basic DTMF Generator Package

```
PackageID: dg (0x0003)
      Version: 1
      Extends: tonegen version 1
      This package defines the basic DTMF tones as signals and
            extends the allowed values of parameter tl of playtone
            in tonegen.
```

E.5.1. Properties

None

E.5.2. Events

None

E.5.3. Signals

dtmf character 0
SignalID: d0 (0x0010)
Generate DTMF 0 tone. The physical characteristic of DTMF 0 is defined in the gateway.
Signal Type: Brief
Duration: Provisioned
Additional Parameters:
None
Additional Values:
d0 (0x0010) is defined as a toneid for playtone

The other dtmf characters are specified in exactly the same way. For brevity's sake only a table with the signal names and the signal IDs is included. Note that each dtmf character is defined as both a signal and a toneid, thus extending the basic tone generation package. Also note that dtmf SignalIds are different from the names used in a digit map.

Signal Name	Signal ID
dtmf character 1	d1 (0x0011)
dtmf character 2	d2 (0x0012)
dtmf character 3	d3 (0x0013)
dtmf character 4	d4 (0x0014)
dtmf character 5	d5 (0x0015)
dtmf character 6	d6 (0x0016)
dtmf character 7	d7 (0x0017)
dtmf character 8	d8 (0x0018)
dtmf character 9	d9 (0x0019)
dtmf character *	ds (0x0020)
dtmf character #	do (0x0021)
dtmf character A	da (0x001a)
dtmf character B	db (0x001b)
dtmf character C	dc (0x001c)
dtmf character D	dd (0x001d)

E.5.4. Statistics

None

E.5.5. Procedures

None

E.6. DTMF detection Package

PackageID: dd (0x0004)
Version: 1
Extends: tonedet version 1
This package defines the basic DTMF tones detection.
This Package extends the possible values of tone id
in the "start tone detected" "end tone detected"
and "long tone detected" events.

Additional tone id values are all tone ids described in package dg (basic DTMF generator package).

The following table maps DTMF events to digit map symbols as described in section 7.1.14.

DTMF;Event Symbol	
d0	;"0"
d1	"1"
d2	"2"
d3	"3"
d4	"4"
d5	"5"
d6	;"6"
d7	;"7"
d8	;"8"
d9	;"9"
da	"A" or "a"
db	"B" or "b"
dc	"C" or "c"
dd	"D" or "d"
ds	"E" or "e"
do	"F" or "f"

E.6.1. Properties

None

E.6.2. Events

DTMF digits
EventIds are defined with the same names as the SignalIds
defined in the table found in section E.5.3

DigitMap Completion Event

```
      EventID: ce, 0x0001
      Raised when a digit map completes as described in section
        7.1.14, or when an AuditValue requesting events is applied to
        a termination, as described in section 7.1.7.

      EventsDescriptor parameters: none.  This event is armed through
        activation of a DigitMap Descriptor in accompaniment with an
        events descriptor specifying the set of DTMF digit events to
        be mapped into the digit map.

      ObservedEventsDescriptor parameters:
            DigitString
                  ParameterID: ds (0x0001)
                  Type: string of digit map symbols (possibly empty)
                  Possible values: any sequence of the characters "0" through
                     "9", "A" through "F", and the long duration modifier "L".
                  Description: the portion of the current dial string as
                    described in section 7.1.14 which matched part or all
                    of an alternative event sequence specified in the digit map.

            Extra
                  ParameterID: ex (0x0002)
                  Type: digit map symbol
                  Possible values: any one of the characters "0" through
                    "9" or "A" through "F"
                  Description: the final digit map symbol added to the current
                    dial string in the case where as a result it was determined
                    that no match of the digit map was possible.  This
                    parameter is present only when that outcome has occurred.

            Termination Method
                  ParameterID:    Meth (0x0003)
                  Type: enumeration
                  Possible values:
                      "MA" (0x0001) Unambiguous match
                      "PM" (0x0002) Partial match, timer expired or
                          unmatched event
                      "FM" (0x0003) Full match, timer expired or
                          unmatched event
                      "AU"  (0x0004) Audit, collection in progress
                  Description: indicates the reason for generation of the event.  See the proced
present only if the "PM" or "FM" termination method is reported.
Its absence in those cases indicates that termination was by timer
expiry.  The "AU" reason shall be used only for a response to an
AuditValue of the termination while digit collection is in progress.
```

E.6.3. Signals

None

E.6.4. Statistics

None

E.6.5. Procedures

None

E.7. Call Progress Tones Generator Package

PackageID: cg, 0x0005
Version: 1
Extends: tonegen version 1
This package defines the basic call progress tones as signals and extends the allowed values of the tl parameter of playtone in tonegen.

E.7.1. Properties

None

E.7.2. Events

None

E.7.3. Signals

Dial Tone
SignaID: dt (0x0030)
Generate dial tone. The physical characteristic of dial tone is available in the gateway.
Signal Type: Timeout
Duration: Provisioned
Additional Parameters:
None
Additional Values
dt (0x0030) is defined as a tone id for playtone

The other tones of this package are defined in exactly the same way. For brevity's sake only a table with the signal names and the signal IDs is included. Note that each tone is defined as both a signal and a toneid, thus extending the basic tone generation package.

Signal Name!Signal ID/tone id Ringing Tone!rt (0x0031) Busy Tone!bt (0x0032) Congestion Tone!ct (0x0033) Special Information Tone!sit(0x0034) Warning Tone!wt (0x0035) Payphone Recognition Tone!pt (0x0036) Call Waiting Tone!cw (0x0037) Caller Waiting Tone!cr (0x0038)

E.7.4. Statistics

None

E.7.5. Procedures

NOTE - The required set of tone ids corresponds to those defined in Recommendation E.180/Q.35 [ITU-T Recommendation E.180/Q.35 (1998)]. See E.180 for definition of the meanings of these tones.

E.8. Call Progress Tones Detection Package

PackageID: cd (0x0006)
Version: 1
Extends: tonedet version 1
This package defines the basic call progress detection tones.
This Package extends the possible values of tone id in the "start tone detected", "end tone detected" and "long tone detected" events.
Additional values
tone id values are defined for start tone detected, end tone detected and long tone detected with the same values as those in package cg (call progress tones generation package).

The required set of tone ids corresponds to Recommendation E.180/Q.35 [ITU-T Recommendation E.180/Q.35 (1998)]. See Recommendation E.180/Q.35 for definition of the meanings of these tones.

E.8.1. Properties

none

E.8.2. Events

Events are defined as in the dtmf detection package (dd) for the tones listed in the table of section E.7.3

E.8.3. Signals

none

E.8.4. Statistics

none

E.8.5. Procedures

none

E.9. Analog Line Supervision Package

PackageID: al, 0x0009
Version: 1
Extends: None
This package defines events and signals for an analog line.

E.9.1. Properties

None

E.9.2. Events

onhook
EventID: on (0x0004)
Detects handset going on hook. If the first event descriptor after (re)establishment of MGC control over the specific termination concerned or the MG as a whole contains the on-hook event request and the line is already on-hook, the MG shall raise an on-hook event as if on-hook had occurred immediately after the event descriptor became active.
EventDescriptor parameters
None
ObservedEventsDescriptor parameters
None
offhook
EventID: of (0x0005)
Detects handset going off hook. If the first event descriptor after (re)establishment of MGC control over the specific termination concerned or the MG as a whole contains the off-hook event request and the line is already off-hook, the MG shall raise an off-hook event as if off-hook had occurred immediately after the event descriptor became active.
EventDescriptor parameters

None
ObservedEventsDescriptor parameters
None
flashhook
EventID: fl, 0x0006
Detects handset flash. A flash occurs when an onhook is followed by an offhook between a minimum and

maximum duration.
EventDescriptor parameters
Minimum duration
ParameterID: mindur (0x0004)
Type: integer in milliseconds
Default value is provisioned
Maximum duration
ParameterID: maxdur (0x0005)
Type: integer in milliseconds
Default value is provisioned
ObservedEventsDescriptor parameters
None

E.9.3. Signals

ring
SignalID: ri, 0x0002
Applies ringing on the line
Signal Type: TimeOut
Duration: Provisioned
Additional Parameters:
Cadence
ParameterID: cad (0x0006)
Type: list of integers representing durations of alternating on and off segments, constituting a complete ringing cycle starting with an on. Units in milliseconds
Default is fixed or provisioned. Restricted function MGs may ignore cadence values they are incapable of generating.
Frequency
ParameterID: freq (0x0007)
Type: integer in Hz
Default is fixed or provisioned. Restricted function MGs may ignore frequency values they are incapable of generating.

E.9.4. Statistics

None

E.9.5. Procedures

None

E.10. Basic Continuity Package

```
PackageID: ct (0x000a)
      Version: 1
      Extends: None
      This package defines events and signals for continuity test.


E.10.1.  Properties

None

E.10.2.  Events

Completion
      EventID: cmp, 0x0005
      Detects test completion.
      EventDescriptor parameters
            None
      ObservedEventsDescriptor parameters
            Result
                  ParameterID: res (0x0008)
                  Type: Enumeration
                  Possible values: success (0x0001), failure (0x0000)


E.10.3.  Signals

Initiate
      SignalID: ini (0x0003)
      Initiates continuity test of a specified type
      Signal Type: OnOff
      Additional Parameters:
            Test type
                  ParameterID: tt
                  Description: this parameter, if present, gives
                    guidance to the MG on which type of continuity
                    test to perform for the indicated Termination.
                    It is necessary only where multiple choices
                    are valid for that termination.  In general,
```



```
                    the MG will have to make use of information
                    provisioned for the given termination in order
                    to know the exact flavour of continuity test
                    to run.
                  Type: enumeration of possible test types
                  Possible Values: q724 (extendible)
                  Default is provisioned
Respond
      SignalID: rsp (0x0004)
      Responds to a continuity test of a specified type
      Signal Type: OnOff
      Additional Parameters:
            Test type
```

ParameterID: tt
Type: enumeration of possible test types
Description: this parameter, if present,
gives guidance to the MG on which type of
continuity test to perform for the
indicated Termination. It is necessary
only where multiple choices are valid for
that termination. In general, the MG will
have to make use of information provisioned
for the given termination in order to know
the exact flavour of continuity test to run.
Possible Values: q724 (extendable)
Default is provisioned

E.10.4. Statistics

None

E.10.5. Procedures

When a MGC initiates or responds to a MG by sending the Initiate or Response signal to the MG, the MG shall perform the continuity test in accordance with the specified test type and additional information provisioned in the MG for the affected termination.

E.11. Network Package

PackageID: nt (0x000b)
Version: 1
Extends: None
This package defines properties of network terminations
independent of network type.

E.11.1. Properties

Maximum Jitter Buffer
PropertyID: jit (0x0007)
This property puts a maximum size on the jitter buffer.
Type: integer in milliseconds
Possible Values: This property is specified in milliseconds.
Defined In: LocalControlDescriptor
Characteristics: read/write

Silence Suppression
PropertyID: ssup (0x1008)
This property indicates whether silence suppression is to
be activated in the encoding of the audio media flow to
which it relates.
Type: Boolean

```
      Possible values: True if silence suppression is to be applied.
      Defined in: LocalControl
      Characteristics: read/write


E.11.2.  Events

network failure
      EventID: netfail, 0x0005
      The termination generates this event upon detection of a
            failure due to external or internal network reasons.
      EventDescriptor parameters
            none
      ObservedEventsDescriptor parameters
            cause
                  ParameterID: cs (0x0001)
                  Type: String
                  Possible values: any text string
                  This parameter may be included with the failure
                  event to provide diagnostic information on the
                  reason of failure.
quality alert
      EventID: qualert, 0x0006
      This property allows the MG to indicate a loss of quality
            of the network connection. The MG may do this by
            measuring packet loss, interarrival jitter, propogation
            delay and then indicating this using a percentage of
            quality loss.
      EventDescriptor parameters
            Threshold
                  ParameterId: th (0x0001)
                  Type: integer
```



```
                  Possible Values: threshold for percent of quality
                        loss measured, calculated based on a
                        provisioned method, that could take into
                        consideration packet loss, jitter, and delay
                        for example.  Event is triggered when
                        calculation exceeds the threshold.
      ObservedEventsDescriptor parameters
            Threshold
                  ParameterId: th (0x0001)
                  Type: integer
                  Possible Values: percent of quality loss measured,
                        calculated based on a provisioned method,
                        that could take into consideration packet loss,
                        jitter, and delay for example.


E.11.3.  Signals

none
```

E.11.4. Statistics

Duration
StatisticsID: dur (0x0001)
Description: Provides duration of time the termination has been in the context.
Type: Double, in milliseconds
Octets Sent
StatisticID: os (0x0002)
Type: double
Possible Values: any 64 bit integer
Octets Received
StatisticID: or (0x0003)
Type: double
Possible Values: any 64 bit integer

E.11.5. Procedures

none

E.12. RTP Package

PackageID: rtp (0x000c)
Version: 1
Extends: Network Package version 1
This package is used to support packet based multimedia data transfer by means of the Real-time Transport Protocol

(RTP) [RFC 1889].

E.12.1. Properties

None

E.12.2. Events

Payload Transition
EventID: pltrans, 0x0001
This event detects and notifies when there is a transition of the RTP payload format from one format to another.
EventDescriptor parameters
none
ObservedEventsDescriptor parameters
ParameterName: rtppayload
ParameterID: rtppltype, 0x01
Type: list of enumerated types.
Possible values: The encoding method shall be specified by using one or several valid encoding names, as defined in the RTP AV

Profile or registered with IANA.

E.12.3. Signals

None

E.12.4. Statistics

Packets Sent
StatisticID: ps (0x0004)
Type: double
Possible Values: any 64 bit integer
Packets Received
StatisticID: pr (0x0005)
Type: double
Possible Values: any 64 bit integer
Packet Loss
StatisticID: pl (0x0006)
Describes the current rate of packet loss on an RTP stream, as defined in IETF RFC 1889. Packet loss is expressed as percentage value: number of packets lost in the interval between two reception reports, divided by the number of packets expected during that interval.
Type: double
Possible Values: a 32 bit whole number and a 32 bit fraction.

Jitter
StatisticID: jit (0x0007)
Requests the current value of the interarrival jitter on an RTP stream as defined in IETF RFC 1889. Jitter measures the variation in interarrival time for RTP data packets.
Delay
StatisticID:delay (0x0008)
Requests the current value of packet propagation delay expressed in timestamp units. Same as average latency.

E.12.5. Procedures

none

E.13. DS0 Package

PackageID: ds0 (0x000d)
Version: 1
Extends: Network Package version 1
This package is used to support DS0 terminations.

E.13.1. Properties

```
Echo Cancellation
      PropertyID: ec (0x0008)
      By default, the telephony gateways always perform
        echo cancellation according to G.165/G.168.
        However, it is necessary, for some calls, to turn
        off these operations.
      Type: enumerated
      Possible Values:
          "on" (echo cancellation is permanently enabled
             and will disregard any incoming tones)
          "g165" (echo cancellation is according to
             G.165/G.168: echo cancellation is on to start
             with, but will turn itself off when it sees a
             2100Hz tone with phase reversal (i.e. a CED fax
             tone, or an ANS modem tone)
          "g164" (echo cancellation is according to G.164)
          "off" (echo cancellation is turned off.)
        The default is "g165".
      Defined In: LocalControlDescriptor
        Characteristics: read/write

Gain Control



Cuervo, Hill, Greene, Huitema, Rayhan, Rosen, Segers          [Page 134]




Internet draft              MEGACO Protocol             February 8, 2000


      PropertyID: gain (0x000a)
      Gain control, or usage of of signal level adaptation and
            noise level reduction is used to adapt the level of
            the signal. However, it is necessary, for example
            for modem calls, to turn off this function.
      Type: enumeration (integer)
      Possible Values:
            The gain control parameter may either be specified
            as "automatic" (0xffffffff), or as an explicit number
            of decibels of gain (any other integer value).
            The default is provisioned in the MG.
      Defined In: LocalControlDescriptor
      Characteristics: read/write


E.13.2.  Events

none

E.13.3.  Signals

none

E.13.4.  Statistics

None

E.13.5.  Procedures
```

None

APPENDIX A EXAMPLE CALL FLOWS (INFORMATIVE)

All Megaco implementors must read the normative part of this document carefully before implementing from it. No one should use the examples in this section as stand-alone explanations of how to create protocol messages.

The examples in this section use SDP for encoding of the Local and Remote stream descriptors. SDP is defined in RFC 2327. If there is any discrepancy between the SDP in the examples, and RFC 2327, the RFC should be consulted for correctness. Audio profiles used are those defined in RFC 1890, and others registered with IANA. For example, G.711 A-law is called PCMA in the SDP, and is assigned profile 0. G.723 is profile 4, and H263 is profile 34. See also http://www.isi.edu/in-notes/iana/assignments/rtp-parameters

A.1. Residential Gateway to Residential Gateway Call

This example scenario illustrates the use of the elements of the protocol to set up a Residential Gateway to Residential Gateway call over an IP-based network. For simplicity, this example assumes that both Residential Gateways involved in the call are controlled by the same Media Gateway Controller.

A.1.1. Programming Residential GW Analog Line Terminations for Idle Behavior

The following illustrates the API invocations from the Media Gateway Controller and Media Gateways to get the Terminations in this scenario programmed for idle behavior. Both the originating and terminating Media Gateways have idle AnalogLine Terminations programmed to look for call initiation events (i.e.-offhook) by using the Modify Command with the appropriate parameters. The null Context is used to indicate that the Terminations are not yet involved in a Context. The ROOT termination is used to indicate the entire MG instead of a termination within the MG.

In this example, MG1 has the IP address 124.124.124.222, MG2 is 125.125.125.111, and the MGC is 123.123.123.4. The default Megaco port is 55555 for all three.

1. An MG registers with an MGC using the ServiceChange command:

```
MG1 to MGC:
MEGACO/1 [124.124.124.222]
Transaction = 9998 {
    Context = - {
```

```
            ServiceChange = ROOT {Services {
                Method=Restart,
                ServiceChangeAddress=55555, Profile=ResGW/1}
            }
        }
    }
```

2. The MGC sends a reply:

```
    MGC to MG1:
    MEGACO/1 [123.123.123.4]:55555
    Reply = 9998 {
       Context = - {ServiceChange = ROOT {
         Services {ServiceChangeAddress=55555, Profile=ResGW/1} } }
    }
```


3. The MGC programs a Termination in the NULL context. The terminationId is A4444, the streamId is 1, the requestId in the Events descriptor is 2222. The mId is the identifier of the sender of this message, in this case, it is the IP address and port [123.123.123.4]:55555. Mode for this stream is set to SendReceive. "al" is the analog line supervision package.

```
    MGC to MG1:
    MEGACO/1 [123.123.123.4]:55555
    Transaction = 9999 {
        Context = - {
            Modify = A4444 {
                Media { Stream = 1 {
                        LocalControl {
                            Mode = SendReceive,
                            ds0/gain=2,   ; in dB,
                            ds0/ec=G165
                        },
                        Local {
    v=0
    c=IN IP4 $
    m=audio $ RTP/AVP 0
    a=fmtp:PCMU VAD=X-NNVAD ; special voice activity
                               ; detection algorithm
                        }
                    }
                },
                Events = 2222 {al/of}
            }
        }
    }
```

The dialplan script could have been loaded into the MG previously. Its function would be to wait for the OffHook, turn on dialtone and start collecting DTMF digits. However in this example, we use the digit map, which is put into place after the offhook is detected (step 5 below).

Note that the embedded EventsDescriptor could have been used to combine steps 3 and 4 with steps 8 and 9, eliminating steps 6 and 7.

4. The MG1 accepts the Modify with this reply:

```
MG1 to MGC:
MEGACO/1 [124.124.124.222]:55555
Reply = 9999 {
   Context = - {Modify = A4444}
```



```
}
```

5. A similar exchange happens between MG2 and the MGC, resulting in an idle Termination called A5555.

A.1.2. Collecting Originator Digits and Initiating Termination

The following builds upon the previously shown conditions. It illustrates the transactions from the Media Gateway Controller and originating Media Gateway (MG1) to get the originating Termination (A4444) through the stages of digit collection required to initiate a connection to the terminating Media Gateway (MG2).

6. MG1 detects an offhook event from User 1 and reports it to the Media Gateway Controller via the Notify Command.

```
MG1 to MGC:
MEGACO/1 [124.124.124.222]:55555
Transaction = 10000 {
   Context = - {
       Notify = A4444 {ObservedEvents =2222 {
         19990729T22000000:al/of}}
   }
}
```

7. And the Notify is acknowledged.

```
MGC to MG1:
MEGACO/1 [123.123.123.4]:55555
Reply = 10000 {
    Context = - {Notify = A4444}
}
```

8. The MGC Modifies the termination to play dial tone, and to look for digits now. There is also an embedded event to stop dialtone upon detection of the first digit. dd is the DTMF Detection package, and ce is the completion event.

```
MGC to MG1:
MEGACO/1 [123.123.123.4]:55555
Transaction = 10001 {
```



```
    Context = - {
        Modify = A4444 {
            Events = 2223 {
                al/on, dd/ce {DigitMap=Dialplan0}
            },
            Signals {cg/dt},
            DigitMap= Dialplan0{
(0| 00|[1-7]xxx|8xxxxxxx|Fxxxxxxx|Exx|91xxxxxxxxxx|9011x.)}
        }
    }
}
```

9. And the Modify is acknowledged.

```
MG1 to MGC:
MEGACO/1 [124.124.124.222]:55555
Reply = 10001 {
    Context = - {Modify = A4444}
}
```

10. Next, digits are accumulated by MG1 as they are dialed by User 1. Dialtone is stopped upon detection of the first digit, using the embedded event in step 8. When an appropriate match is made of collected digits against the currently programmed Dialplan for A4444, another Notify is sent to the Media Gateway Controller.

```
MG1 to MGC:
MEGACO/1 [124.124.124.222]:55555
Transaction = 10002 {
   Context = - {
       Notify = A4444 {ObservedEvents =2223 {
         19990729T22010001:dd/ce{ds="916135551212",Meth=FM}}}
   }
}
```

11. And the Notify is acknowledged.

```
MGC to MG1:
MEGACO/1 [123.123.123.4]:55555
Reply = 10002 {
    Context = - {Notify = A4444}
}
```


12. The controller then analyses the digits and determines that a connection needs to be made from MG1 to MG2. Both the TDM termination A4444, and an RTP termination are added to a new context in MG1. Mode is ReceiveOnly since Remote descriptor values are not yet specified. Preferred codecs are in the MGC's preferred order of choice.

```
MGC to MG1:
MEGACO/1 [123.123.123.4]:55555
Transaction = 10003 {
    Context = $ {
       Add = A4444,
       Add = $ {
           Media {
             Stream = 1 {
                  LocalControl {
                      Mode = ReceiveOnly,

                      nt/jit=40, ; in ms
                  },
                  Local {
v=0
c=IN IP4 $
m=audio $ RTP/AVP 4
a=ptime:30
v=0
c=IN IP4 $
m=audio $ RTP/AVP 0
                  }
              }
          }
       }
    }
}
```

NOTE - The MGC states its preferred parameter values as a series of sdp blocks in Local. The MG fills in the Local Descriptor in the Reply.

13. MG1 acknowledges the new Termination and fills in the Local IP address and UDP port. It also makes a choice for the codec based on the MGC preferences in Local. MG1 sets the RTP port to 2222.

```
MEGACO/1 [124.124.124.222]:55555
Reply = 10003 {
   Context = 2000 {
```



```
      Add = A4444,
      Add=A4445{
         Media {
             Stream = 1 {
                 Local {
v=0
c=IN IP4 124.124.124.222
m=audio 2222 RTP/AVP 4
a=ptime:30
a=recvonly
                 } ; RTP profile for G.723 is 4
             }
         }
      }
   }
}
```

14. The MGC will now associate A5555 with a new Context on MG2, and establish an RTP Stream (i.e, A5556 will be assigned), SendReceive connection through to the originating user, User 1. The MGC also sets ring on A5555.

```
MGC to MG2:
MEGACO/1 [123.123.123.4]:55555
Transaction = 50003 {
    Context = $ {
       Add = A5555  { Media {
            Stream = 1 {
                 LocalControl {Mode = SendReceive} }},
            Events = 1234{al/of},
            Signals {al/ri}
            },
       Add  = $ {Media {
            Stream = 1 {
                 LocalControl {
                    Mode = SendReceive,
                    nt/jit=40 ; in ms
                 },
                 Local {
v=0
c=IN IP4 $
m=audio $ RTP/AVP 4
a=ptime:30
                 },
                 Remote {
v=0
```

```
     c=IN IP4 124.124.124.222
     m=audio 2222 RTP/AVP 4
     a=ptime:30
                        } ; RTP profile for G.723 is 4
                    }
                }
            }
        }
     }
```

15. This is acknowledged. The stream port number is different from the control port number. In this case it is 1111 (in the SDP).

```
     MG2 to MGC:
     MEGACO/1 [124.124.124.222]:55555
     Reply = 50003 {
        Context = 5000 {
           Add = A5555{},
           Add = A5556{
              Media {
                 Stream = 1 {
                     Local {
     v=0
     c=IN IP4 125.125.125.111
     m=audio 1111 RTP/AVP 4
     }
                 } ; RTP profile for G723 is 4
              }
           }
        }
     }
```

16. The above IPAddr and UDPport need to be given to MG1 now.

```
     MGC to MG1:
     MEGACO/1 [123.123.123.4]:55555
     Transaction = 10005 {
       Context = 2000 {
         Modify = A4444 {
           Signals {cg/rt}
         },
         Modify = A4445 {
            Media {
```

```
                Stream = 1 {
                    Remote {
    v=0
    c=IN IP4 125.125.125.111
    m=audio 1111 RTP/AVP 4
                    }
                } ; RTP profile for G723 is 4
            }
        }
      }
    }

    MG1 to MGC:
    MEGACO/1 [124.124.124.222]:55555
    Reply = 10005 {
       Context = 2000 {Modify = A4444, Modify = A4445}
    }
```

17. The two gateways are now connected and User 1 hears the RingBack. The MG2 now waits until User2 picks up the receiver and then the two-way call is established.

```
    From MG2 to MGC:

    MEGACO/1 [125.125.125.111]:55555
    Transaction = 50005 {
       Context = 5000 {
           Notify = A5555 {ObservedEvents =1234 {
             19990729T22020002:al/of}}
       }
    }

    From MGC to MG2:

    MEGACO/1 [123.123.123.4]:55555
    Reply = 50005 {
        Context = - {Notify = A5555}
    }

    From MGC to MG2:

    MEGACO/1 [123.123.123.4]:55555
    Transaction = 50006 {
       Context = 5000 {
          Modify = A5555 {
             Events = 1235 {al/on},
```

```
            Signals { } ; to turn off ringing
         }
      }
   }
```

From MG2 to MGC:

```
MEGACO/1 [125.125.125.111]:55555
Reply = 50006 {
 Context = 5000 {Modify = A4445}
}
```

18. Change mode on MG1 to SendReceive, and stop the ringback.

MGC to MG1:

```
MEGACO/1 [123.123.123.4]:55555
Transaction = 10006 {
   Context = 2000 {
      Modify = A4445 {
         Media {
            Stream = 1 {
               LocalControl {
                  Mode=SendReceive
               }
            }
         }
      },
      Modify = A4444 {
         Signals { }
      }
   }
}
```

from MG1 to MGC:

```
MEGACO/1 [124.124.124.222]:55555
Reply = 10006 {
   Context = 2000 {Modify = A4445, Modify = A4444}}
```

E. The MGC decides to Audit the RTP termination on MG2.

```
MEGACO/1 [123.123.123.4]:55555
Transaction = 50007 {
   Context = - {AuditValue = A5556{
      Audit{Media, DigitMap, Events, Signals, Packages, Statistics }}
```

```
      }
   }
```

20. The MG2 replies. An RTP termination has no events nor signals, so these are left out in the reply .

```
MEGACO/1 [125.125.125.111]:55555
Reply = 50007 {
   Context = - {
       AuditValue = A5556 {
          Media {
             Stream = 1 {
                 LocalControl { Mode = SendReceive,
                    nt/jit=40 },
                 Local {
v=0
c=IN IP4 125.125.125.111
m=audio 1111 RTP/AVP  4
a=ptime:30
                },
                 Remote {
v=0
c=IN IP4 124.124.124.222
m=audio 2222 RTP/AVP  4
a=ptime:30
                 } } },
          Packages {nt-1, rtp-1},
          Statistics { rtp/ps=1200,  ; packets sent
                       nt/os=62300, ; octets sent
                       rtp/pr=700, ; packets received
                       nt/or=45100, ; octets received
                       rtp/pl=0.2,  ; % packet loss
                       rtp/jit=20,
                       rtp/delay=40 } ; avg latency
       }
    }
}
```

21. When the MGC receives an onhook signal from one of the MGs, it brings down the call. In this example, the user at MG2 hangs up first.

From MG2 to MGC:

```
MEGACO/1 [125.125.125.111]:55555
Transaction = 50008 {
   Context = 5000 {
       Notify = A5555 {ObservedEvents =1235 {
          19990729T24020002:al/on}
       }
   }
}

From MGC to MG2:

MEGACO/1 [123.123.123.4]:55555
Reply = 50008 {
    Context = - {Notify = A5555}
}
```

22. The MGC now sends both MGs a Subtract to take down the call. Only the subtracts to MG2 are shown here. Each termination has its own set of statistics that it gathers. An MGC may not need to request both to be returned. A5555 is a physical termination, and A5556 is an RTP termination.

```
From MGC to MG2:

MEGACO/1 [123.123.123.4]:55555
Transaction = 50009 {
   Context = 5000 {
      Subtract = A5555 {Audit{Statistics}},
      Subtract = A5556 {Audit{Statistics}}
   }
}

From MG2 to MGC:

MEGACO/1 [125.125.125.111]:55555
Reply = 50009 {
   Context = 5000 {
     Subtract = A5555 {
          Statistics {
             nt/os=45123, ; Octets Sent
             nt/dur=40 ; in seconds
             }
       },
       Subtract = A5556 {
          Statistics {
             rtp/ps=1245, ; packets sent
```



```
             nt/os=62345, ; octets sent
             rtp/pr=780, ; packets received
             nt/or=45123, ; octets received
```

```
                rtp/pl=10, ;  % packets lost
                rtp/jit=27,
                rtp/delay=48 ; average latency
            }
        }
    }
}
```

23. The MGC now sets up both MG1 and MG2 to be ready to detect the next off-hook event. See step 1. Note that this could be the default state of a termination in the null context, and if this were the case, no message need be sent from the MGC to the MG. Once a termination returns to the null context, it goes back to the default termination values for that termination.